HELSINGIN TEKNILLINEN KORKEAKOULU

Tietotekniikan Osasto

Teemu Ikonen

Logistiikan tietojärjestelmän välityspalvelimen toteutus tilakoneilla

Diplomityö

TEKNILLINEN KORKEAKOULU Tietotekniikan osasto		DIPLOMITYÖN TIIVISTELMÄ	
Tekijä	Teemu Ikonen 44023A	Päiväys	7.12.2004
		Sivumäärä	74
Työn nimi	Logistiikan tietojärjestelmän välityspalvelimen toteutus tilakoneilla		
Professuuri	Ohjelmistojärjestelmät	Koodi	T-106
Työn valvoja	Prof. Jorma Tarhio		
Työn ohjaaja	Petri Sumu TietoEnator Oyj		

Hajautetussa tietojärjestelmässä prosessien välisellä kommunikoinnin toteutuksella on ratkaiseva merkitys koko järjestelmän toiminnan ja luotettavuuden kannalta. Kommunikointi voidaan toteuttaa lukuisilla tavoilla, joista yksi on on välityspalvelin, johon kaikki tietojärjestelmän prosessit ovat yhteydessä. Välityspalvelimen vastuulla on prosessien lähettämien viestien luotettava välitys, sekä niiden jakaminen palvelinten kesken.

Diplomityössä kehitettiin välityspalvelin, joka pystyy huolehtimaan suurenkin järjestelmän viestiliikenteestä luotettavasti. Välityspalvelimen arkkitehtuuri perustuu komento-tilakone suunnittelumallin, jolla voidaan yhdellä suoritussäikeellä hallita asynkronisesti useita samanaikaisia viestitransaktioita. Työssä tarkastellaan myös ongelmia joita toteutuksessa ja tuotannossa syntyi ja kuinka ne ratkaistiin tai kierrettiin.

Välityspalvelin otettiin tuotantokäyttöön ja nykyisellään sen odotettu elinikä on tämän vuosikymmenen loppuun. Järjestelmällä on yli 100 päivittäistä käyttäjää.

Avainsanat

Välityspalvelin, Tilakone, 3-taso arkkitehtuuri

HELSINKI UNIVERSITY OF TECHNOLOGY Department of Computer Science and Engineering		ABSTRACT OF MASTER'S THESIS	
Author	Teemu Ikonen 44023A	Date	7.12.2004
		Pages	74
Title of thesis	Implementation of Middleware software for logistic data system with state machine pattern		
Professorship	Software Systems	Professorship Code	T-106
Supervisor	Dr. Jorma Tarhio		
Instructor	Petri Sumu TietoEnator Ltd.		

Distributed systems require a stable and reliable communication media. One implementation of this requirement is a system based on messaging over a middleware server, where all distributed components are connected. Middleware server is responsible of the reliable message delivery and the integrity of the messages.

This master thesis consists of design and development of middleware system that can handle messaging of a real distributed system. Middleware architecture is based on command driven state pattern, where is possible to run separate parallel operations on single execution thread. Theses covers also problem solving and solutions both in development and production.

Middleware system is currently on production and its expected life time is to the end of this decade. Production system that middleware runs has over hundred daily users.

Keywords

Middleware, State machine, 3-tier architecture

Alkulause

Olen tehnyt diplomityöni analyytikkona TietoEnator Oyj:lle Espoossa. Työssä tehty välityspalvelin ohjelmisto on tehty osana Pro2000 projektia, asiakkaana Steveco Oy.

Kiitän kaikkia työhön osallistuneita TietoEnator Oyj:ssä, jotka tekivät tämän työn mahdolliseksi ja tukivat kehitystyötä. Erityinen kiitos kuuluu myös työni valvojalle professori Jorma Tarhiolle.

Espoo 7.12.2004

Teemu Ikonen

Komeentakuja 3C 32

02210 Espoo

Sähköposti: teemu.ikonen@iki.fi

Puhelin: +358 40 7035561

Sisällysluettelo

Alkulause 4
Termistö 7
1 Johdanto 8
1.1 Yritykset 9
2 Ympäristö 10
2.1 Satama 10
2.1.1 Prosessit 10
2.1.2 Vasteaikavaatimukset 11
2.1.3 Sovellukset 11
2.2 Laitteistoympäristö 12
2.3 Valvontaratkaisut 12
3 Järjestelmäarkkitehtuuri 13
3.1 2-taso arkkitehtuuri 13
3.2 3-taso arkkitehtuuri 14
4 Vaatimukset 17
4.1 Käytettävyys 17
4.2 Vikasietoisuus 18
4.3 Suorituskyky 19
4.4 Sovellusliittymät 20
5 Suunnittelu 22
5.1 Käytettävyys ja Vikatilanteet 32
5.2 Suorituskyky ja Rajoitukset 33
5.2.1 Sovellusten hajautus 35
5.3 Tilakoneet 35
5.4 Kaupalliset vaihtoehdot 42
5.5 Prosessivalvonta 43
5.5.1 Raportointi 44
5.6 Hallintakäyttöliittymä 44
5.7 Kannettavuus 46
5.8 Yhteenveto 47
6 Toteutus 48
6.1 Välityspalvelin 48
6.2 Liittymäkirjastot 50
6.3 Prosessinvalvonta 52
6.4 Hallintaliittymä 54
6.5 Toteutuksen ongelmia 55
6.6 Yhteenveto 61
7 Testaus 62
7.1 Suunnitelma 62
7.2 Toteutus 62
7.2.1 Testausohjelmistot 63
7.3 Yhteenveto 64
8 Tuotanto 66
8.1 Suunnitelma ja Toteutus 66
8.2 Liittyminen valvontaratkaisuihin 66
8.3 Tuotanto-ongelmat 67
9 Yhteenveto 71

Lähdeluettelo 73

Taulukot

Termistö 7
Sovellusten tilat 24
Palveluviestin tilakone 37
Hallintaviestin tilakone 38
Tilat 39
Tapahtumat 40
Ehdot 41
Välityspalvelimen komponentit 49
Sovellusliittymäkomponentit 51
Prosessinvalvonnan komponentit 52
Hallintaliittymän komponentit 55
Palvelumatriisi 59
Matriisin Täyttymisesimerkki 59
Lukkiuman Havainnointi 60
Testausohjelmistot 64

Kuvat

2-arkkitehtuuri, järjestelmärakenne 13
3-arkkitehtuuri, järjestelmärakenne 15
Komponentit 22
Monisäikeinen arkkitehtuuri 23
Yksisäikeinen arkkitehtuuri 25
Perusviestivuo 30
Monimutkainen viestivuo 30
Hallintakäyttöliittymä 45
Sovellusliittymän UML-kaavio 50
Lukkiumatilanne 1 57
Lukkiumatilanne 2 58
Testausohjelmiston pääikkuna 63
Virheilmoitus käyttäjälle 67

Termistö

Tässä kappaleessa esitellään käytetyt termit. Monille termeille on olemassa jo vakiintunut suomenkielinen vastine, mutta useat ovat vielä vähemmän tunnettuja ja ohjelmistotekniikan alalla käytetäänkin luontevasti englanninkielistä versiota. Tässä työssä on pyritty käyttämään aina suomenkielistä termiä, kun sellainen on ollut käytettävissä. Joidenkin termien yhteydessä käytetään muistutuksena suluissa englanninkielistä alkuperäistä termiä silloin, kun termin suomenkielinen vastine ei ole vakiintunut tai luonteva, jolloin tekstin ymmärtäminen vaikeutuisi merkittävästi.

Taulukossa 1 on lueteltu termit sisältäen englanninkielinen versio viitteeksi. Suurin osa termeistä on tarkistettu ATK-sanakirjan mukaan [ATK 03].

Termi	*Selite*	*Englanniksi*
Järjestelmä	Järjestelmä, jota varten välityspalvelin tehtiin.	
Instanssi	Sovelluksen ajossa oleva kopio tai olion muistissa oleva kopio.	Instance
Korkea Käytettävyys	Korkea vikasietoisuus	High Availability
Lukkiuma	Säikeiden lukitustilanne, jossa säikeiden omistamat lukot estävät kummankin säikeen etenemisen.	Deadlock
Luonnosohjelma	Ohjelman rakenteen esittäminen yksinkertaistetulla koodilla.	Pseudocode
Säie	Sovelluksessa itsenäinen ajossa oleva konteksti	Thread
Vastake	Ohjelmoinnissa käytettävä kahva TCP-yhteyteen.	Socket
Suppea Asiakas	Asiakassovellus, joka jättää ohjelmistologiikan pääosin palvelimen tehtäväksi	Thin Client
Osaava Asiakas	Asiakassovellus, joka toteuttaa ohjelmistologiikan suurimmaksi osaksi itse.	Fat Client
Välityspalvelin	Tapahtumamonitori-ohjelmisto, joka välittää viestejä komponenttien välillä.	Middleware

Taulukko 1Termistö

LYHENNELUETTELO

TCP	Transmit Control Protocol
DNS	Domain Name Service
ISDN	Integrated Services Digital Network
HTTP	Hypertext Transfer Protocol
JDK	Java™ Development Kit
URL	Uniform Resource Locator

1 Johdanto

Sataman toiminta perustuu optimoituun logistiikkaketjuun, jonka tehokkuudesta riippuu sen tuottavuus. Sataman liiketoimintaideana on tavaran jakelu, huolinta ja välivarastointi. Riippuen alueesta tarvitaan hyvin erilaisia prosesseja, joiden täytyy kuitenkin nivoutua yhteen saumattomasti niin johtamisen, tiedonkulun kuin suorittamisen tasolla.

Tietotekniikka nähdään alueella välineenä, jolla logistiikkaketjua voidaan tehostaa ja sen läpivientiä nopeuttaa. Tietoteknisien ratkaisujen täytyy tukea koko arvoketjua ja auttaa yksinkertaisesti tuottamaan lisää voittoa yritykselle.

Perinteisesti logistiikan tietojärjestelmät ovat olleet suuria ja raskaita järjestelmiä, joita on voitu hallita keskitetysti keskuspalvelinympäristössä. Tähän kehitykseen johti aikanaan malli jossa jokaisella käyttäjällä oli etäkäyttöterminaali yhteen keskuskoneeseen, jossa kukin käyttäjä on ajanut instanssia yhdestä ohjelmasta. Ohjelmisto on sitten ottanut yhteyden jaettuun tietokantaan, jossa kaikki toimintaan tarvittava tieto on talletettu suhteellisen jalostamattomassa muodossa. Arkkitehtuuri tunnetaan yleisnimellä Osaava Asiakas. [Prim 95]

Osaavaan asiakkaaseen perustuva arkkitehtuuri on toiminut hyvin niin kauan kun ohjelmistojen koko ei ole kasvanut liian suureksi ja niiden toiminnallisuus on pysynyt selkeästi erikoistuneena. Ongelmia on syntynyt, kun vaadittu toiminnallisuus on kasvanut monimutkaisemmaksi ja tiedon esitystä on pyritty jalostamaan entisestään. Tällöin keskuskone joutuu käyttämään laskentatehoaan epäedullisesti; yhä monimutkaisempiin käyttöliittymiin ja tiedon jalostamiseen käyttäjäystävälliseen muotoon. Samaan aikaan työasemat kehittyivät tehokkaiksi, mutta niiden resurssit jäivät hyödyntämättä. Keskitetty arkkitehtuuri kärsii myös skaalautuvuusongelmista sekä teknisessä että taloudellisessa mielessä, koska keskitettyä ratkaisua ei voi hajauttaa helposti useaan pienempään keskuskoneeseen. Tuotannon kasvaessa on täytynyt aina päivittää keskuspalvelin suurempaan. Laitteistovalmistajien liiketoimintamallista johtuvista syistä suurien palvelinten hinta tulee usein selvästi suuremmaksi käyttäjää kohden, kuin esimerkiksi työasemien kustannukset. Tämä johtuu siitä, että keskuspalvelimien päivitysaskeleet ovat diskreettejä, joten jokainen päivitys seuraavalle asteelle on erittäin kallista.

Henkilökohtaisten Microsoft Windows-pohjaisten työasemien tehojen kasvu 90-luvulla mahdollisti asiakassovelluksen ajamisen keskuskoneen sijaan kunkin käyttäjän omassa työasemassa. Siirtämällä osaava asiakassovellus henkilökohtaiselle työasemalle jää keskuskoneen vastuulle vain tietokanta, näin saatiin siirrettyä merkittävä osa järjestelmän kuormasta työasemille. Ilmeinen seuraava pullonkaula suorituskyvylle arkkitehtuurissa on kuitenkin edelleen keskuskone, joka ajaa tietokantaa. Jokainen sovellus joutuu käyttämään tietokantaa varsin raskaasti, koska tieto on tietokannassa täysin jalostamattomassa muodossa. Näin ollen päädytään täysin samaan ongelmaan kuin keskitetyssä ratkaisussa, vaikkakin laitteistoinvestoinnit ovatkin huomattavasti kustannustehokkaammassa käytössä.

Ratkaisuksi ongelmaan ryhdyttiin esittämään välityspalvelimiin perustuvia arkkitehtuureja, [Prim 95]. jotka muistuttavat periaatteeltaan reitittimiä. Välityspalvelin-arkkitehtuurissa toisella puolen ovat suppeat asiakassovellukset, joiden ainoa rooli on esittää mahdollisimman käyttäjäystävällisen liittymä järjestelmään. Arkkitehtuurin toinen osa taas koostuu yksittäisistä palvelinohjelmistoista, jotka kukin huolehtivat vain yhden eristetyn tehtävän täyttämisestä. Arkkitehtuurissa yhden palvelinsovelluksen rooli voisi olla esimerkiksi autentikointi ja konfiguraatio, toisen palvelimen rooli loki-tiedostojen analyysi ja kolmannen logistiikkaoperaation palveleminen. Välityspalvelinarkkitehtuuri perustuu sanomanvälitykseen ja transaktioihin eli tapahtumiin. Sanomat on määritelty järjestelmään ja asiakassovellukset voivat kommunikoida palvelinsovellusten kanssa sanomilla käyttäen välityspalvelinta yhteyspisteenään. Välityspalvelin päättää mikä palvelin saa minkäkin sanoman ja priorisoi tapahtumien järjestyksen. Välityspalvelinarkkitehtuurissa tietokannan rasitus on kevyempää, koska vain palvelinten tarvitsee olla yhteydessä tietokantaan, samaten jokainen sovellus varaa

resursseja vain hetken vuorollaan, eikä jatkuvasti kuten Osaaviin sovelluksiin perustuvissa arkkitehtuureissa. Sanomat on määriteltävissä vapaasti kuhunkin tarkoitukseen ja ne voidaan optimoida aina tarpeen mukaan. Näin ollen voidaan minimoida asiakassovelluksen tarve käyttää sanomia. Arkkitehtuuri tunnetaan nimellä Suppea Sovellus.

Vuonna 1998 Tieto Oyj (Nyt TietoEnator Oyj) voitti tarjouskilpailun Steveco Oy Pro2000 hankkeesta, jonka tarkoitus oli uusia vanhat 80-luvulta peräisin olevat osaaviin sovelluksiin perustuvat tietojärjestelmät. Yhtenä mahdollisena järjestelmän toteuttamismenetelmänä harkittiin välityspalvelin-arkkitehtuuria, jossa järjestelmän toiminta perustuu sanomille. Tämä työ käsittelee välityspalvelimen suunnittelua, toteutusta ja tuotantokäyttöönottoa Pro2000 projektissa.

1.1 Yritykset

TietoEnator Oyj on konserni, joka toimittaa tietotekniikkapalveluja mm. metsäteollisuudelle ja logistiikan tarpeisiin. Erityisesti logistiikassa pyritään luomaan ratkaisuja, jotka tukevat mahdollisimman tehokkaasti asiakkaiden liiketoimintaa ja integroituvat niihin saumattomasti. [Tieto 2001]

Steveco Oy on konserni johon kuuluu yhtenä osana satamatoiminta sitä tukevine liiketoiminta-alueineen. Yrityksen toiminta-ajatuksena on suomen teollisuuden ja kaupan, erityisesti metsäteollisuuden, viennin ja tuonnin, sekä kauttakulkuliikenteen hoitaminen ja kehittäminen kannattavasti asiakkaiden kilpailukykyä edistävällä tavalla. Tämä käsittää satama-, terminaali-, informaatio- ja kokonaiskuljetuspalvelut. [Stev 01]

2 Ympäristö

Kappale esittelee ympäristön johon ohjelmisto tehtiin sekä siellä vaikuttavista tekijöistä, joilla oli merkitystä ohjelmiston suunnittelussa.

Satama on tavaraliikenteen solmukohta maa- ja meriliikenteen välillä. Tärkeimmät toiminnot ovat konttiliikenne, sellu- ja paperiliikenne, bulkkitavara ja transit-toiminta. Transit-toiminnassa tullaus lisää monimutkaisuutta, koska satamalla on lakisääteisiä velvoitteita tavaran koskemattomuuden ja tullauksen osalta.

Satama toimii vuoden ja vuorokauden ympäri, työrytmin katkaisevat vain lyhyet ammattiliiton valvomat ruoka- ja kahvitauot, jolloin kaikki normaali toiminta satama-alueella pysähtyy. Sääolosuhteet ja sesonkien mukaan vaihtuvat kuljetustarpeet vaativat joustavaa prosessiketjua jossa tietotekniikan aiheuttamia katkoksia ei sallita.

2.1 Satama

Sataman liikenne jakautuu karkeasti konttiliikenteeseen, sellu- ja paperiliikenteeseen, bulkkitavaraan ja transittoimintaan.

Konttiliikenne kostuu kontteihin pakatuista tavarasta, jota voidaan käsitellä yhtenä loogisena yksikkönä. Kontteja myös varastoidaan täytenä ja tyhjänä satama-alueelle joista syntyykin satamalle tunnusomainen ulkonäkö pitkine konttiriveineen. Jokaisella kontilla on yksilöllinen numero, josta sen kulkua voidaan maailmalla seurata. Konttivarustamot omistavat kontit vaikka ne olisivat kenen tahansa käytössä, satamat ovat velvollisia seuraamaan ja raportoimaan kunkin kontin kauttakulun.

Sellu- ja paperiliikenne on Suomalaiselle satamalle tyypillistä, paperirullat ja raakasellu lastataan rahtialuksiin lähinnä Keski-Eurooppaan vietäväksi. Paperirullat matkustavat joko rekoilla konteissa tai junalla paperitehtaalta satamaan, jossa ne puretaan ja toimitetaan laivoihin.

Bulkkitavara on kuivaa irtotavaraa joka useimmiten tarkoittaa lannoitteita, hiiltä tai muita teollisuuden raaka-aineita.

Transittoiminta tarkoittaa tavaroiden pidemmälle vietyä käsittelyä, purkamista, pakkaamista, jatkokuljetusta, jakelua, tulliselvityksiä ja kylmäsäilytystä. Suurin osa maahan tuotavista tuotteista, kuten autot, kuuluvat transittoiminnan piiriin.

2.1.1 Prosessit

Sataman tavaraliikenteen jokaisella muodolla on oma käsittelyprosessinsa jota noudatetaan tarkasti. Nämä prosessit pyrkivät luomaan mahdollisimman pienin resurssein joustavan tavaraliikenteen joissa poikkeukset, kuten epämääräiset sääolosuhteet, eivät vaikuta toimintaan merkittävästi. Nykyisellään käytettävät prosessit eivät enää olisi mahdollisia ilman tietotekniikkaa ja sen reaaliaikaista tiedonvälitystä.

Prosessien tehoa optimoitaessa tarvitaan mittareita ja erottelukykyä käsittelyprosessin eri vaiheisiin. Vaatimus on ongelmallinen, koska työntekijät eivät välttämättä ole korkeasti koulutettuja ja monimutkaisen termistön ja ohjelmistojen kouluttaminen työntekijöille ei ole taloudellisesti kannattavaa. Kenttä-olosuhteissa tietoteknisen laitteiden käyttömahdollisuudet voivat olla hyvin rajoitettuja tai muutoin hankalia, esimerkiksi kovalla pakkasella meriolosuhteissa ei voida käyttää pitkään kannettavaa terminaalia ilman hansikkaita. Käyttöliittymien tulee olla täysin käsittelyprosessia tukevia, yksinkertaisia, mahdollisimman virheenkorjaavia ja poikkeustilanteissa intuitiivisia.

Tietotekniikan osalta välityspalvelin-arkkitehtuurissa palvelinsovellukset ja itse välityspalvelin ta-

kaavat luotettavuuden ja käsittelyprosessin sujuvuuden, jos jokin palvelu ei vastaa annetussa ajassa tai se vikaantuu voidaan käyttää rinnakkaispalvelinta. Asiakassovelluksessa tämä voidaan joko piilottaa tai pyytää käyttäjää toistamaan toimenpide. Parhaassa tapauksessa käyttäjä ei huomaa muuta kuin hieman tavanomaista pidemmän vasteajan.

2.1.2 Vasteaikavaatimukset

Käsittelyprosessin jouheva suoritus edellyttää että tietyt toimenpiteet käsittelyprosessissa eivät vie enemmän kuin niille varattua aikaa. Esimerkiksi kun rekka ajaa sataman portista sisään, pitää tieto lastikirjasta olla jo huolintapisteessä valmiina kun rekka saapuu sinne. Kellontarkalla suorituksella ja toimenpiteiden optimoinnilla voidaan tavaraliikenteen läpivienti maksimoida ja käsittelyprosessiin varattu aika voidaan minimoida. Mikäli prosessia optimoimalla saadaan paperitehtaan ja sataman väliä ajava rekka saadaan ulos satamasta mahdollisimman nopeasti, on se mitattavissa selvänä liikevoittona kun samana päivänä voidaan ottaa sisään yksi rekka enemmän. Esitetyssä ympäristössä ohjelmistoilta vaaditaan luonnollisesti nopeaa vasteaikaa. Tässä yhteydessä on kuitenkin tärkeää ottaa huomioon että vasteajan tulisi olla mahdollisimman vakio, tämä vähentää kenttäolosuhteissa virhekäytön riskiä. Rutinoitunut käyttäjä oppii nimittäin nopeasti painamaan seuraavaan ruutuun vievää näppäintä katsomatta lopputulosta, tämä voi johtaa sekaannuksiin jos näppäintä painetaankin väärällä hetkellä.

Järjestelmän vasteajoilla ja palvelun laadulla on psykologisesti tärkeä merkitys työntekijöille ja samalla se luo parempaa kuvaa sataman asiakkaalle palvelun laadusta. Nopeilla ohjelmistoilla voidaan myös minimoida aika joka tarvitaan ohjelmiston käyttöön keskittymiseen, jolloin käyttäjän huomiokykyä ei varata kokonaan.

2.1.3 Sovellukset

Teollisuusympäristönä satama jakaantuu karkeasti toimistotyöntekijöihin ja kenttätyöntekijöihin.

Tyypilliset sovellukset jakautuvat kahteen luokkaan; työasemilla ajettavat Microsoft Windows-alustan monimutkaiset hallintosovellukset ja kannettavilla päätteillä ajettavat käsittelyprosessia tukevat sovellukset.

Toimistotyöntekijät käyttävät tavallisia pöytätyöasemia joilla ajetaan asiakassovelluksia tilausten sisäänsyöttöön, raportointiin, seurantaan, taloushallintoon ja poikkeustilanteiden hallintaan.

Kenttätyöntekijät käyttävät teollisuuspäätteitä, jotka ovat usein kannettavia päätteitä satama-alueen omassa radioverkossa [LXE]. Kannettavissa päätteissä ajetaan sovelluksia jotka ovat käsittelyprosessin suorittamisen tukena kuten varastohallinta, tavaran ulos- ja sisäänkirjaaminen, poikkeustilanteiden hallinta ja tavaraliikenteen seurannan tarkastuspisteiden kirjaaminen.

Hallintosovellukset ovat raskaita sovelluksia joissa on painotettu paljon käyttäjäystävällisyyteen ja tehokkuuteen. Ohjelmistoja käytetään usein paperilla olevan datan syöttämiseen järjestelmään, jolloin niiden rakenteelta vaaditaan samaa järjestystä kuin papereilla olevalla tiedolla.

Kannettavat päätesovellukset ovat pääasiassa prosessia tukevia kirjausohjelmisto joilla prosessin tapahtumat ja mahdolliset poikkeukset merkitään ylös. Kannettavia päätesovelluksia käytetään usein myös korjaamaan mahdollisesti muilla sovelluksilla tehtyjä virheitä. Tyypillisin esimerkki lienee väärin syötetty kontin numero joka joudutaan korjaamaan tietojärjestelmään heti sen tullessa satamaan. Kannettavia päätesovelluksia ajetaan HP-UX ympäristössä.

Molemmat sovellustyypit toimivat välityspalvelimen sanomilla ja käyttävät samoja sanomia. Näin voidaan palvelinohjelmistot toteuttaa vain kerran huolimatta kahdenlaisista asiakassovelluksista.

2.2 Laitteistoympäristö

Laitteistoympäristössä on kolmenlaisia erilaisia laitteistoja. Palvelinohjelmistoja itse välityspalvelinta varten on käytössä kaksi Hewlett-Packard moniprosessoripalvelinta joista toinen oli testauskäytössä sekä ylimääräisenä varakoneena tuotannon vikaantumisen varalta. Palvelinten prosessoriarkkitehtuuri on PA-RISC, eli redusoituun käskykantaan perustuva prosessori. Hallinnon asiakassovelluksia varten on n. 200 työasemaa varustettuna Microsoft Windows NT 4.0. käyttöjärjestelmällä. Kannettavia palvelinsovelluksia varten on LXE-radioterminaaleja joita on käytössä on n. 40 [LXE].

Sataman lähiverkko perustuu perinteiseen IP [RFQ791] verkkoon joten luonnollinen ratkaisu viestiliikenteen toteutukseen on TPC [RFQ793] protokolla palvelinten ja asiakassovellusten välillä.

2.3 Valvontaratkaisut

Järjestelmää valvotaan etävalvontana TietoEnator Oyj:n Imatran valvontakeskuksesta. Valvonta käsittää tuen ja ongelmatilanteiden havainnoinnin, hoidon sekä tiedottamisen. Valvonta pystyy havaitsemaan tyypillisimmät ongelmat automaattisesti, useimmiten valvontakeskuksen päivystäjä saa tiedon vikaantumisesta ja ottaa yhteyttä Stevecon järjestelmähallitsijaan ennen kuin varsinaiset käyttäjät ehtivät raportoida ongelmasta tai edes havaita sitä.

Välityspalvelimen valvontaan riittää yksinkertainen sovellus jota ajetaan minuutin välein. Sovellus tekee pyynnön välityspalvelimelle, joka palauttaa yksinkertaisen tilastoraportin, mikäli välityspalvelin pystyy palauttamaan tilastoraportin välityspalvelimen katsotaan olevan toimintakunnossa.

Välityspalvelimen osana tehtiin myös valvontasovellus, joka on kuitenkin jätetty suurilta osin tämän työn ulkopuolelle. Valvontasovellus on käytännössä ohjelmisto joka pystyy käynnistämään ja tarkkailemaan käyttöjärjestelmässä ajettavia prosesseja, jos prosessi ei enää vastaa tarkastusviestiin se voidaan päättää ja käynnistää uudestaan. Tieto vikaantumisesta voidaan välittää sähköpostitse järjestelmän ylläpitäjälle.

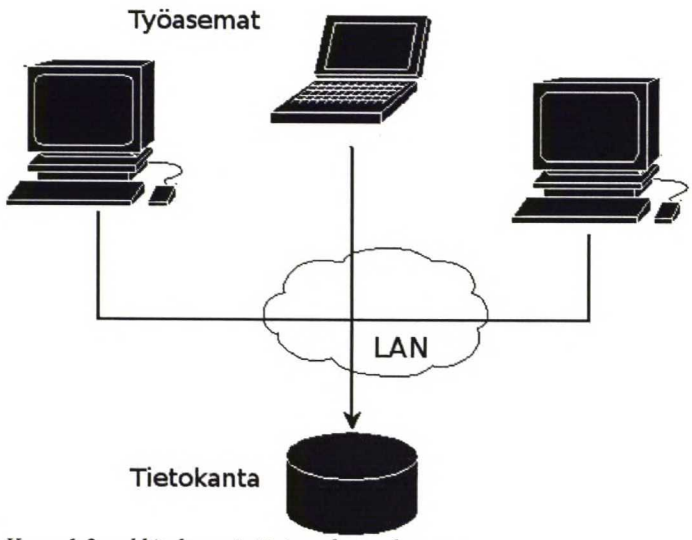
3 Järjestelmäarkkitehtuuri

Kappale esittelee koko sen järjestelmäarkkitehtuurin jota varten välitysohjelmisto rakennettiin, se myös esittelee kaksi arkkitehtuurivaihtoehtoa joita harkittiin järjestelmää suunniteltaessa. Suunnittelussa päädyttiin 3-tasoarkkitehtuuriin.

Järjestelmän arkkitehtuuri perustuu nykyisellään kolmitasoiseen malliin, jossa järjestelmä jakaantuu asiakassovelluksiin, välityspalvelimeen sekä palvelinsovelluksiin. Palvelinsovellukset ovat yhteydessä tietokantaan jossa järjestelmän tieto talletetaan. Palvelinsovellukset ovat yhteydessä tietokantaan ja välityspalvelimeen. Asiakassovellukset ovat yhteydessä vain välityspalvelimeen.

3.1 2-taso arkkitehtuuri

Perinteisesti yritysten tietojärjestelmät on suunniteltu 2-taso arkkitehtuurilla, jossa kaiken logiikan sisältävä osaava asiakas on suorassa yhteydessä relaatiotietokantaan. Yhteys tietokantaan on näissä järjestelmissä toteutettu hyvin alhaisella tasolla, kuten SQL-kielellä [ISO9075], eikä tietokanta ymmärrä järjestelmän tallettamien tietojen semanttista merkitystä. Palvelintasolla, eli tietokannassa, ei siten ole minkäänlaista sovelluskohtaista älykkyyttä. Kuvassa 1 on esitetty 2-taso arkkitehtuurin perustuvan järjestelmän yleiskuva.



Kuva 1 2-arkkitehtuuri, järjestelmärakenne

Ratkaisun ilmeisiä etuja ovat sovelluksen selkeys ja itseriittoisuus, kaikki logiikka on vain yhdessä paikassa joten sovellus tarvitsee vain tietokantayhteyden toimiakseen. Ohjelmistoa on myös suhteellisen helppo ylläpitää kun kaikki muutokset ja korjaukset voidaan tehdä vain yhteen ohjelmaan, joka on samalla käyttöliittymä ja itse järjestelmälogiikan (eng. "business-logic") sisältävä komponentti.

Logiikan sijoittaminen yhteen paikkaan on kuitenkin samalla arkkitehtuurin suurin ongelma, kaiken logiikan ollessa asiakaspuolella tiedon käsittely käy tietokannalle työlääksi. Asiakassovellukset joutuvat toimimaan samalla tavalla ja prosessoimaan samat tiedot käyttäen tietokantaan talletettua raakatietoa, jolloin tietokannan kuorma kasvaa helposti erittäin suureksi. Ongelma pahenee mitä jalostamattomammassa muodossa tieto on kantaan talletettu, jokainen asiakassovellus joutuu tekemään lukuisia pyyntöjä tietokantaan yksinkertaistenkin asioiden käsittelemiseksi. Siirrettävät tietomäärät ovat suuria käyttöliittymän tarpeeseen nähden, koska relaatiotietokannan rajapinta antaa vain rajoi-

tetusti mahdollisuuksia tiedon esijalostamiseen. Usein toiminto jota ei tarvitsisi suorittaa moneen kertaan joudutaan kuitenkin toistamaan jokaisessa asiakassovelluksessa, koska sovellukset eivät ole yhteydessä toisiinsa.

Asiakassovellusten järjestelmähallinta on ongelmallista, koska systeemin ylläpitäjällä ei käytännössä ole mitään kontrollia itse järjestelmään. Mitään keskitettyä järjestelmäähän ei ole, vaan jokaisen käyttäjän työpöydällä on vain oma instanssi järjestelmästä. Ainoa yhteinen nimittäjä on tietokanta. Päivitystarpeen edessä, tai kun ohjelmasta löytyy virhe, joudutaan aina raskaaseen ylläpitoprosessiin jossa jokaisen käyttäjän työasema pitää erikseen päivittää. Ongelmaa voidaan kiertää asentamalla asiakassovellus jaetulle verkkolevylle, mutta tämä toimii vain jos toimipisteiden organisaatio tukee tällaista asennusta. Päivitys joudutaan myös tekemään yhtenä suurena päivityksenä jolloin tietokanta joudutaan ottamaan alas, näin varmistetaan että kaikki käyttäjät ajavat varmasti samaa versiota sovelluksesta.

2-taso arkkitehtuuriin perustuvan järjestelmän skaalautuvuus on vaikeasti toteutettavissa, koska ainoa mahdollisuus on tietokantapalvelimen tehon kasvattaminen. Tietokantapalvelimen päivittäminen käy helposti kustannustehottomaksi kun tätä pullonkaularesurssia ei voi kasvattaa asteittain vaan joudutaan ostamaan suurempi tietokantapalvelin sekä uusimaan tietokannan lisenssit. Yhdessä nämä muodostavat suuren kustannuserän.

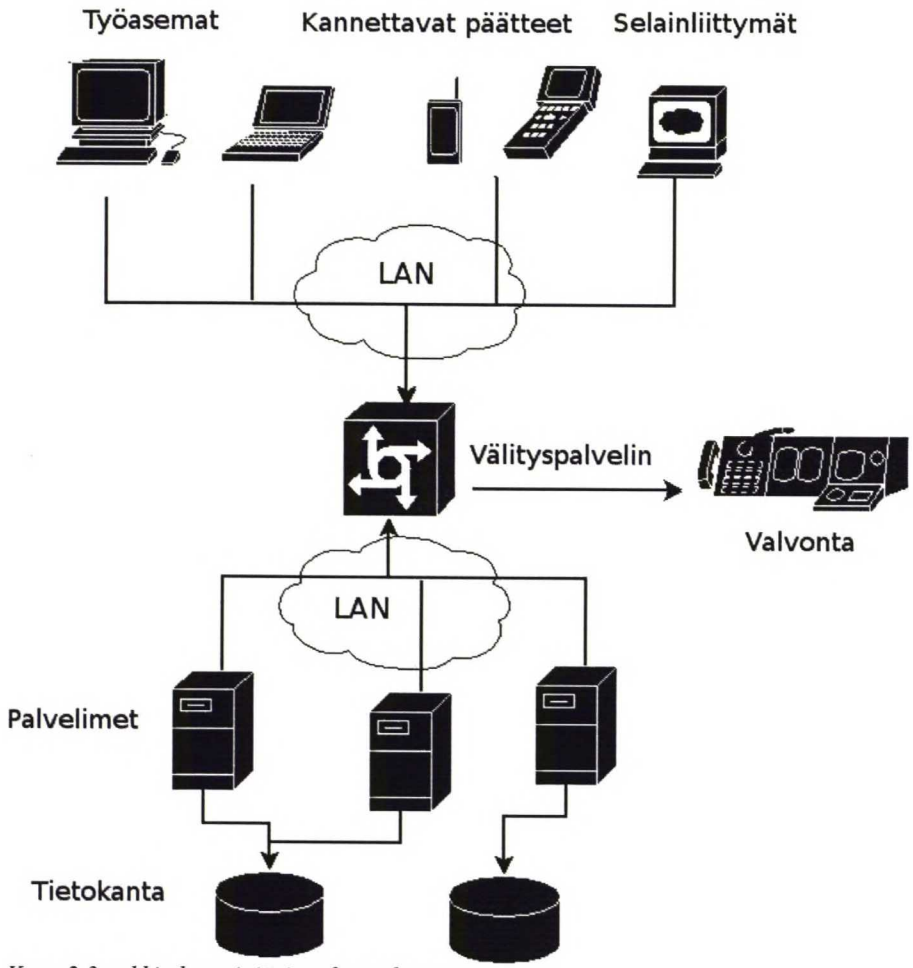
Tiedon sekä resurssien jakaminen on ongelmallista toteuttaa, sillä yhtäaikaista pääsyä tietokantaan on hyvin vaikea kontrolloida. Relaatiotietokannat tarjoavat toiminnallisuutta synkronisen tiedonsaannin varmistamiseksi, mutta käytännössä relaatiotietokannan taulujen lukittaminen on vaikeaa tehdä oikein. Järjestelmän suorituskyky kärsii kun useat sovellukset yrittävät päästä yhtä aikaa samaan tietoon käsiksi.

Mikäli asiakassovelluksia tarvitaan useammalle erilaiselle alustalle käy ylläpito haasteelliseksi, koska samat muutokset pitää aina tarjota samaan aikaan eri alustoille. Kahdenkin eri alustan tuki samalle ohjelmistolle vaatii tarkkaa versiokuria. Testaus on myös haasteellista sillä asiakassovelluksessa on huomattava määrä järjestelmän älykkyydestä, joten on todennäköistä että ne vikaantuvat käytössä. Vikaantumiset näkyvät tavalliselle käyttäjälle käyttökatkoksina ja vaativat usein käyttäjää käynnistämään ohjelman uudestaan. Virhetilanteista ei tyypillisesti jää mitään jälkiä, koska käyttäjät eivät ota virheilmoituksia ylös eivätkä raportoi ongelmista, kun yksinkertainen uudelleenkäynnistys korjaa tilanteen. Järjestelmän laatu ei parane nopeasti, koska viat eivät tule korjatuksi.

Yhteenvetona sanotaan että 2-taso arkkitehtuuriin perustuvaa järjestelmää on vaikeaa hajauttaa, tietokanta tulee lopulta väistämättä pullonkaulaksi tietokannan transaktiomäärien noustessa.

3.2 3-taso arkkitehtuuri

Järjestelmien koon kasvaessa ja niiden monimutkaistuessa siirryttiin 2-taso arkkitehtuureista 3-taso arkkitehtuureihin, joissa asiakassovellus on mahdollisimman suppea. Arkkitehtuurin filosofiana on sijoittaa businesslogiikka yhteen paikkaan ja sitten tuottaa suppeita asiakassovelluksia eri tarpeisiin. Kommunikaatiota businesslogiikkakomponentin ja asiakassovellusten välillä hoitaa välityspalvelin jota kutsutaan myös tapahtumamonitoriksi. Välityspalvelin tarjoaa sanomapohjaisen mallin jossa välityspalvelimeen yhteydessä olevat sovellukset kommunikoivat lähettämällä toisilleen sanomia. Sanomaliikenne asiakassovelluksien ja palvelimien välillä on kevyttä, koska niiden välillä kulkevat viestit ovat korkealle jalostettuja ja suodatettua. Kuvassa 6 on esitetty 3-taso arkkitehtuurilla rakennetun järjestelmän rakenne.



Kuva 2 3-arkkitehtuuri, järjestelmärakenne

Mallin etuna on hallittavuus ja skaalautuvuus. Järjestelmää voidaan valvoa yhdessä paikassa, eikä asiakassovellusten vikaantumiset vaikuta merkittävästi sen toimintaan. Tietokantaresurssien jako on helpompaa, kun pääsy tietoon ja lukitukset voidaan hoitaa yhdessä paikassa tai ainakin paljon yksinkertaisemmin kuin tilanteessa, jossa jokaisella asiakassovelluksella on samanarvoinen pääsy tietokantaan.

Palvelinprosesseja voidaan hajauttaa suhteellisen helposti jos niissä esiintyy pullonkauloja. Kaikkien palvelinsovellusten ollessa yhteydessä samaan tietokantaan, on tietokanta käytännössä edelleen pullonkaularesurssi, mutta sen käyttö on huomattavasti kustannustehokkaampaa kuin 2-taso arkkitehtuurissa. Järjestelmän skaalautuvuus onkin käytännössä merkittävästi parempi. Suuremmissa järjestelmissä on suhteellisen helppoa partitioida eri tiedot eri tietokantakoneisiin käyttäen jakokriteerinä sitä, kuinka paljon palvelimet käyttävät toisiaan ristiin. Palvelinsovelluksethan pääsevät helposti toistensa palveluihin käsiksi viestimällä välityspalvelimen kautta. Perustietoja tarjoavat palvelinsovellukset, jotka eivät käytä muita palveluja, voidaan ensimmäisenä siirtää eri kantaan mikäli tähän ilmenee tarvetta. Suorituskykyvaatimuksen kasvaessa palvelimia voidaan edelleen hajauttaa, periaatteessa on mahdollista asentaa jokainen palvelin käyttämään eri kantakonetta. Välityspalvelimen nopeus muodostaa lopullisen pullonkaulan.

Järjestelmän päivitykset ovat suoraviivaisia kunhan sovelluslogiikka pysyy samana. Välityspalvelin tukee prosessia, jossa palvelinsovellus päivitetään ja käynnistetään päivitetystä versiosta uusi instanssi ja lopetetaan vanha instanssi. Käyttäjä ei huomaa minkäänlaista katkosta järjestelmässä.

Palvelinsovelluksissa on suurin osa älykkyydestä, joten myös todennäköistä että ne vikaantuvat useimmiten. 3-taso arkkitehtuurin etu on, että nämä viat voidaan piilottaa melko helposti joko pitämällä kahta instanssia samasta palvelinsovelluksesta, tai sitten käynnistämällä se heti uudestaan. Palvelinsovellusten vikaantuessa tai muuten vikaantuessa tiedostojärjestelmään jää loki ja mahdollisesti muistivedos-tiedosto, (eng. "core image") josta kehittäjät voivat suhteellisen suoraviivaisesti selvittää mikä ongelman aiheutti.

Asiakassovellus on mallissa kevyt ja yksinkertainen, joten se on suhteellisen helppo testata. Asiakassovelluksia varten voidaan rakentaa simuloituja palvelimia, joiden avulla testaus on helpottuu. Palvelinsovellusten automaattinen testaus on mahdollista käyttäen simuloitua viestiliikennettä.

Seuraamalla viestiliikennettä voidaan selvittää järjestelmän suorituskyvyn pullonkaulat. Viestien tilastoista nähdään selkeästi mitä palvelinta kuormitetaan eniten tai mitä viestiä käytetään eniten. Hitaille viesteille voidaan käynnistää lisää rinnakkaisia palvelimia jakamaan kuormaa. Välityspalvelimen valvontaohjelmisto voidaan myös asettaa seuraamaan tilastoja ja hälyttämään, mikäli jonkin viestin viive tai palvelimen jonon pituus kasvaa liian suureksi.

Yhteenvetona sanotaan että 3-taso arkkitehtuuriin perustuvaa järjestelmää on skaalattava ja hyvin hallittava, mutta sen toteuttaminen vaatii merkittävästi enemmän ympäristön ja verkon suunnittelua kuin 2-taso arkkitehtuurin.

4 Vaatimukset

Kappale esittelee välityspalvelin-ohjelmistolle esitetyt merkittävimmät suunnitteluun vaikuttaneet toiminnalliset ja ei-toiminnalliset vaatimukset, sekä esittää syyt näille vaatimuksille. Vaatimukset muotoutuivat lopulliseen asuunsa suurelta osin vasta suunnitteluvaiheessa, joten kappaleessa sivutaan myös suunnittelua.

Tavoitteena oli tehdä välityspalvelin joka olisi erittäin luotettava ja ennen kaikkea vikasietoinen. Ohjelmisto ei saisi missään tilanteessa turmella tietoa jota se välittää, koska sanomien tiedon eheys on järjestelmässä erittäin tärkeää. Käsittelyprosessit riippuvat siitä, että tieto saadaan siirrettyä täsmällisesti ja ennen kaikkea eheänä. Tuotantokatkokset eivät ole läheskään yhtä vaarallisia kuin tiedon vääristyminen, täten ohjelman ei käytettävyysaste ei tarvinnut olla korkeasti käytettävä, jossa käyttöaika ei saa alittaa tiettyä prosenttia vuodesta. [Marcus 00], sivu 10 .

Yleistyksenä voidaan todeta että vaatimukset olivat hyvin perinteiset ja realistiset. Mitään ohjelmistoa ei voida saada 100-prosenttisen varmaksi [Marcus 00], mutta vian ilmetessä se ei saa toimia väärin huomaamatta ja vian selvityksen pitää olla mahdollisimman nopeaa. Ohjelmiston täytyy tietysti myös tyydyttää tietyt muut minimikriteerit ollakseen käytettävä, sen täytyy olla tarpeeksi vakaa ja nopea.

4.1 Käytettävyys

Käytettävyydestä puhuttaessa ei tässä yhteydessä keskitytä käytettävyyteen sen perinteisessä muodossa eli ohjelmiston käyttöliittymään. Käytettävyys tarkoittaa välitysohjelmistolle luotettavuutta ja sen sopivuutta kyseiseen tehtävään [Marcus 00].

Räätälöidyissä järjestelmissä ohjelmiston vaatimukset laadusta eivät ole lähelläkään vastaavan tuotteen tasoa, joten yleensä testitapaukset ja laadunvarmistus on yleensä hyvin rajoitettuja keskittyen vain niihin tapauksiin joissa ohjelmistoa tullaan käyttämään. Laadunvarmistuksessa merkittävin rajoite on budjetti, joka heijastuu suoraan aikatauluun ja varsinkin testausresurssien määrään. Ohjelmisto testataan melko suppealla määrällä testitapauksia ja rajoitettuun tuotantoon voidaan siirtyä hyvinkin keskeneräisellä ohjelmistolla. Palvelinohjelmistojen laatu asettaa näin haasteen välitysohjelmistolle, jonka avulla vianselvityksen täytyy olla mahdollisimman nopeaa.

Välityspalvelimeen perustuvassa arkkitehtuurissa, jossa välitetään viestejä asiakassovellusten ja palvelinsovellusten välillä, vianselvitys on luontevaa aloittaa viestiliikenteestä. Testauksen näkökulmasta tällainen järjestelmä on musta laatikko [Tamres 02] jonne lähetetään viesti ja oletetaan, että se ohjautuu oikealle palvelimelle ja vastausviesti saadaan takaisin mikäli palvelin sen pystyy tuottamaan. Välityspalvelimen sijainti on hyvä ongelmien selvitykseen, koska kun viestiprotokolla on helposti luettavissa, on suhteellisen helppoa vetää johtopäätöksiä analyysitiedoston sisällöstä ja päätellä kummassa päässä vika on, asiakas- vai palvelinsovelluksessa.

Välityspalvelin täytyikin suunnitella ja toteuttaa siten, että välitettävä tieto on ihmisen luettavissa olevaa selkokielistä tekstiä ja merkkejä, joita voi tarkastella tavallisella tekstipohjaisella päätteellä tai tekstinkäsittelyohjelmalla. Käytännössä vaatimus tarkoittaa sitä, että viestiliikenteessä käytetään vain merkkejä jotka voidaan esittää jollain tyypillisellä kirjaimella (eng, "font"), kuten Microsoft Windows käyttöjärjestelmän standardikirjaimella Courierilla.

Viestien jäljitykseen vaaditaan, että välityspalvelinta voidaan käskeä kirjoittamaan milloin ja minkä tahansa käyttäjän, viestin tai palvelinsovelluksen viestiliikenne tiedostoon aikaleimattuna, josta se on analysoitavissa jälkeenpäin. Jäljitystiedoston nimi määräytyy käytetyn suodatuskriteerin mukaan ja se on joko käyttäjän, viestin tai palvelinsovelluksen nimen pohjalta rakennettu.

Tarvittiin myös välityspalvelimen testausohjelma, jolla voitaisiin vian toistamiseksi lähettää jälji-

tystiedostoon kirjoitettu viesti välityspalvelimelle. Jäljitystiedostojen kanssa yhteensopiva ohjelma nopeuttaa merkittävästi vian selvitystä, kun ei tarvitse luottaa esimerkiksi käyttäjän puhelimitse antamaan vikailmoitukseen.

Testausohjelman lisäksi tarvittiin myös graafinen tai vaihtoehtoisesti komentorivipohjainen hallintakäyttöliittymä, jossa voitaisiin nähdä yhdellä silmäyksellä palvelinsovellusten, viestien ja asiakassovellusten tilat.

Välityspalvelimen toteuttaessa esitetyt vaatimukset vianselvitysprosessi voidaan toteuttaa seuraavasti.

1. Käyttäjä ilmoittaa että jokin toiminto ei toimi odotetusti.
2. Järjestelmävalvoja asettaa välityspalvelimen valvontaohjelmistolla käyttäjän jäljityksen päälle.
3. Järjestelmävalvoja pyytää käyttäjää toistamaan toiminnon
4. Järjestelmävalvoja ottaa pois jäljityksen ja ryhtyy tutkimaan tiedoston sisältämiä viestejä.

Järjestelmänvalvoja voi myös toimittaa tiedoston palvelinsovelluksen lokitiedoston ohella kehittäjille jotka pystyvät testausohjelmistolla lähettämään viestit uudestaan ja näin simuloida tilanteen testausympäristössä.

Valvontaohjelmiston piti tukea yksinkertaista tilastoraportointia joskaan mitään erityisiä vaatimuksia ei esitetty. Myöhemmin kävi ilmi, että komentorivipohjainen hallintaohjelma keskimääräisen kuorman ja palvelinten terveyden analysointiin osoittautui riittäväksi, eikä graafista käyttöliittymää tarvittu.

4.2 Vikasietoisuus

Järjestelmää suunniteltaessa oli alusta lähtien ilmeistä, että palvelinsovellusten laatu olisi aluksi kyseenalainen, testaus ei voisi annetuilla resursseilla olla täysin kattavaa ja johtuen käytetystä ohjelmointiympäristöstä palvelinsovellusten vikaantumiset olivat hyvin yleisiä. Palvelinohjelmistot toteutettiin C++ kielellä [ISO14882], jossa muistinsuojauksen puuttumisen takia tapaturmaiset ohjelmointivirheet ovat hyvin yleisiä. [Marcus 00] viittaa sivulla 12 IEEE:n tutkimukseen ja esittää, että 30% järjestelmän tuotantokatkoksista johtuvat palvelimien virheistä, 40% tuotantokatkoksista ollessa suunniteltuja. Tästä voidaan päätellä, että suunnittelemattomista tuotantokatkoksista 50% johtuu palvelinten virheistä.

Palvelinsovelluksen vikaantuminen voi johtaa useaan erilaiseen virhetilanteeseen. Järjestelmä suunniteltiin siten, että kaikki sen välittämät viestit ovat atomisia; mikäli viestiä ei saada välitettyä eheänä asiakassovellukselta palvelinsovellukselle, sitä ei lähetetä ollenkaan. Välityspalvelimen täytyy vastaanottaa viesti onnistuneesti, ennen kuin se voidaan välittää eteenpäin. Viestien atomisella käsittelyllä vältytään ongelmilta, joita syntyy kun joko asiakas- tai palvelinsovellus vikaantuu tai niihin menetetään yhteys. Välityspalvelimen sovellusrajapinta tehtiin noudattamaan samaa logiikka, eli sovellusrajapinta palautti virheen sovellukselle ellei se saanut viestiä kokonaan lähetettyä tai vastaanotettua. Atominen viestien välitys vaati kuitenkin rajoittamaan viestien koon johonkin enimmäiskokoon, koska välityspalvelin voi pitää muistissaan vain rajallisen määrän viestejä yhtä aikaa.

Useimmissa tapauksissa palvelinsovellus vikaantui siten, että käyttöjärjestelmä lopetti sen suorituksen. Tämän ohella toinen tyypillinen ongelma oli palvelimen jumiutuminen siten, että se ei joko vastaanottanut viestejä tai sitten ei vastannut niihin annetun aikarajan sisällä. Jumiutuminen johtui useimmiten virheestä joka aiheutti päättymättömän silmukan josta ohjelma ei koskaan poistunut. Mikäli silmukka varasi jatkuvasti muistia, johti tilanne ennen pitkää lopettamiseen prosessille varatun muistin loppuessa tai valvontaohjelman huomatessa, että kyseinen palvelinsovellus ei enää vas-

tannut viesteihin. Välityspalvelimeen täytyi siis tehdä asetuksiin määriteltävä aikaraja jonka aikana kukin viesti piti pystyä käsittelemään, käytännössä oikean arvon löytäminen tälle aikarajalle osoittautui kuitenkin yllättävän vaikeaksi.

Välityspalvelimen piti tukea useita saman palvelinsovelluksen instansseja, tällä tavalla välityspalvelin pystyy ohjaamaan viestit toiselle palvelinsovellukselle mikäli ensimmäinen on varattu tai mikäli se on poissa käytöstä. Valvontaohjelmisto hoiti vikaantuneen palvelimen uudelleenkäynnistäminen.

Ensisijainen tavoite vikasietoisuudessa oli piilottaa ongelmatilanne käyttäjältä siten, että jos palvelinsovellus vikaantui, pyrittiin viesti ohjaamaan toisella samaa viestiä ymmärtävälle palvelimelle. Virhetilanne, jossa palvelinprosessi ehti käsittelemään viestin muttei pystynyt enää lähettämään vastausviestiä asiakassovellukselle, osoittautui ongelmalliseksi. Välityspalvelimen välitettyä viestin palvelinsovellukselle välityspalvelin ei voi aikarajan umpeutuessa tietää miksi palvelinsovellus ei vastannut viestiin. Mikäli vastausta viestiin ei kuulu, ei välitysohjelmisto voi yksinkertaisesti lähettää viestiä uudestaan, koska viestillä on voinut olla pysyviä vaikutuksia järjestelmään. Otetaan esimerkiksi tilanne jossa asiakassovellus on lähettänyt viestin, jossa sitä pyydetään tuottamaan laskutustapahtuma. Palvelinsovellus on saanut viestin mutta ei vastaa siihen aikarajan sisällä, oletetaan myös että tämä johtuu hetkellisestä kuormapiikistä palvelimessa eikä virheestä ohjelmassa. Mikäli välityspalvelin nyt lähettäisi viestin uudelleen, se loisi järjestelmään uuden laskutustapahtuman jokaista uudelleenlähetystä kohti. Tämän pohjalta oli paras valita toimintamalli, jossa virhe vain raportoidaan asiakassovellukselle joka esittää sen selkeästi erottuvalla viestiruudulla. Näin käyttäjälle jää lopullinen päätöksenteko ja hän on myös tietoinen, että edellinen suoritettu operaatio on voinut epäonnistua. Käyttäjä voi myös raportoida virhetilanteen tukipalveluun.

4.3 Suorituskyky

Järjestelmää suunniteltaessa oli ilmeistä, että välitysohjelmiston suorituskyky ei tulisi olemaan merkittävä ongelma. Välityspalvelin-arkkitehtuurissa viestit voidaan optimoida, joten niissä ei useimmiten tarvitse kuljettaa merkittäviä määriä tietoa, näin ollen viestit eivät tavukooltaan ole keskimäärin suuria. Viestien sisältämällä tiedolla voi olla korkea jalostusaste verrattuna raakatietoon tietokannassa, ainoastaan raportointiprosessien viestiliikenteessä oli tarvetta selkeästi keskimääräistä suuremmalle viestikoolle. Välityspalvelimen käsittely viesteille on kevyttä, koska sen tarvitsee käytännössä ymmärtää vain viestin otsakkeet sen ohjaamiseksi oikealle palvelimelle. Välityspalvelimen ei näin ollen tarvitse purkaa ja analysoida koko viestiä, joten sille ei ole juurikaan merkitystä kuinka suuri viesti on.

Viestin määrä sekunnissa on välityspalvelimelle suorituskykykriteerinä epärelevantti. Asiakassovelluksien lähettämille viesteille pitää aina saada vastausviesti, joten järjestelmän suorituskykyä rajoittaa palvelinsovelluksien keskimääräinen nopeus. Näin ollen päädyttiin transaktiopohjaiseen määrittelyyn, jossa sadan yhtäaikaisen, 50 kilotavua kooltaan olevan viestin hallinta olisi riittävä suorituskyky välityspalvelimelle. Käytännössä vaadittu transaktio-suorituskyky tarkoittaa sitä, että välityspalvelimessa on yhtä aikaa auki sata transaktiota, eli enintään sata asiakassovellusta on lähettänyt viestin sadalle eri palvelinsovellukselle odottaen niihin vastausta. Järjestelmässä on käytössä vain muutama kymmenen palvelinsovellusta joten suorituskyvystä ei näin ollen tullut ongelmia. Yhtäaikaisten transaktioiden vaatima määrä on myös niin pieni, ettei viestien käsittelyyn tarvita mitään erityisiä algoritmeja. Välitysohjelmiston suunnittelun ja toteutuksen painopiste voitiin siis pitää luotettavuudessa; monimutkaisia optiomointeja ei tarvinnut toteuttaa.

Rajoittava tekijä on se, kuinka nopeasti palvelinsovellukset pystyvät vastaamaan asiakassovelluksien pyyntöihin. Kukin transaktio varaa välityspalvelimesta resurssin niin kauan, kunnes palvelinsovellus vastaa viestillä. Suorituskyvyn rajoitin on siten yhtäaikaisten asiakassovellusten määrä.

4.4 Sovellusliittymät

Järjestelmän asiakassovellusalustaksi valittiin Microsoft Windows NT 4.0 käyttöjärjestelmä ja ohjelmointiympäristöksi Microsoft Visual Studio C++. Asiakassovelluksia tehtiin myös Visual Basic-ohjelmointiympäristöllä [Balena 99] mutta näin tehdyt sovellukset rajoittuivat lähinnä Microsoft Excel-makroihin, jossa viesteillä haettiin palvelinprosesseilta tietoja, joilla sitten täytettiin lomake raportointia varten. Palvelinprossien alusta on HP-UX 10.20 ympäristö ja ohjelmointiympäristöksi Hewlett-Packardin toimittama C++ kääntäjä.

Ympäristön IP-pohjainen lähiverkko saneli käytännössä siirtorajapinnaksi TCP protokollan. TCP protokollalla on myös molemmissa ohjelmointiympäristöissä kypsynyt verkko-ohjelmointirajapinta, näin se oli kaikin puolin paras valinta. Vaihtoehtona esitettiin nimettyihin putkiin perustuva rajapinta, mutta tämä huomattiin nopeasti käyttökelvottomaksi, koska tämä tekniikka ei ole HP-UX ja Windows NT käyttöjärjestelmissä yhteensopivaa.

Asiakas- ja palvelinsovellusten sama ohjelmointikieli mahdollisti sovellusrajapinnan ohjelmoimisen siten, että se oli helposti käännettävissä kummallekin ympäristölle ilman mitään suuria muutoksia. Sovellusrajapinta rakennettiin siten, että sekä asiakas- että palvelin puolella pystyttiin käyttämään mahdollisimman paljon samoja luokkia. Helpoiten tämä onnistui suunnittelemalla oliorajapinta jossa vain muutaman olion tarvitsi olla erityisesti asiakas- ja palvelinsovellukselle tarkoitettu. Sovellusrajapinta toteutettiin kummassakin ohjelmointiympäristössä jaetulla kirjastolla, joka tarjoaa sovellukselle oliorajapinnan, jolla luodaan istunto välityspalvelimeen ja käsitellään viestejä. Rajapinta tarjoaa toiminnallisuuden myös virhetilanteiden käsittelyyn, raportointiin sekä niistä toipumiseen.

Asiakassovellus luo istunnon instantioimalla istunto-luokan, jonka luontiparametrina se käyttää välityspalvelimen sijainnin URL osoitteen [RFC1738]. Istunto-olio hakee ympäristöstä käyttäjän nimen sekä muista tarvittavia tietoja. Palvelinsovellus luo istunnon samoin kuin asiakassovellus mutta se käyttää luontiparametrina lisäksi listaa viesteistä joita se haluaa palvella. Välityspalvelin voi sitten käyttää tätä tietoa päättääkseen mille palvelimelle viesti pitää ohjata. Sovelluksilla täytyy olla yksiselitteinen nimi, mutta usea palvelinsovellus voi mainostaa palvelevansa samaa viestiä.

Sovellusrajapinnalla käsiteltävät viestit koostuvat yhdestä otsakesegmentistä sekä sovellussegmenteistä joita edustavat oliot rajapinnassa. Otsakesegmentti on aina vakiomuotoinen ja sen rakenne on sisäänrakennettu sovellusrajapintaan. Otsake sisältää viestin nimen jonka avulla välityspalvelin päättää mille palvelimelle viesti pitää ohjata. Sovellussegmentit ovat vapaasti sovelluksen määriteltävissä eikä sovellusrajapinta välitä niiden sisällöstä, näin sovellusrajapinta sallii täysin vapaamuotoisen viestin jossa asiakas- ja palvelinsovellus tietävät viestin nimestä yhteisellä sopimuksella mitä sovellussegmenttejä se sisältää. Sovellussegmentit voidaan määritellä siten, että ne sisältävät tietoja joita voidaan käyttää useamman eri viestin osana, esimerkiksi käyttäjätietoa tarvitaan useissa erilaisissa viesteissä.

Segmentit koostuvat kentistä joihin sovellusrajapinnassa viitataan indeksillä, segmentin ensimmäiset kentät on varattu segmentin nimelle sekä kenttien lukumäärälle. Sovellussegmentit toteutetaan perimällä perussegmenttiluokka ja toteuttamalla luokkaan metodit, jotka hakevat tiedon käyttäen kovakoodattua indeksiä. Viestin otsakesegmentti sisältää viestissä olevien sovellussegmenttien määrän muttei koko viestin pituutta. Segmenteissä ei myöskään ole kenttien pituustietoa vaan ainoastaan segmentissä olevien kenttien lukumäärä.

Sovellus rakentaa viestin segmentti kerrallaan, asettaa otsake-segmenttiin viestin nimen ja lähettää sen istunnon metodilla. Välityspalvelin reitittää viestin palvelinsovellukselle, jossa sovellusrajapinta lukee otsakesegmentin ja palauttaa sovellussegmentit. Palvelinsovellus käsittelee viestin segmentti kerrallaan, sen kuitenkin täytyy tietää mitä segmenttejä viestissä on ja pyytää niitä nimeltä. Palvelinsovellus rakentaa sitten segmentti kerrallaan vastausviestin ja lähettää sen takaisin välityspalvelimelle joka ohjaa viestin asiakassovellukselle. Asiakassovellus purkaa viestin samalla tavalla kuin

palvelinsovellus, segmentti kerrallaan.

Sovellusrajapinta on tehty kestämään käyttökatkokset, jos esimerkiksi yhteys välityspalvelimeen menetetään, istunto-olio yrittää säännöllisesti uudelleen yhteyttä välityspalvelimeen, joten sovelluksen ei tarvitse toteuttaa paljoakaan virhekäsittelyä. Viestiliikenteeseen liittyvät virheet raportoidaan kuitenkin heti, näitä ovat esimerkiksi virheet joissa palvelinsovellus ei vastaa viestiin tai viestiä palvelevaa palvelinsovellusta ei löydy.

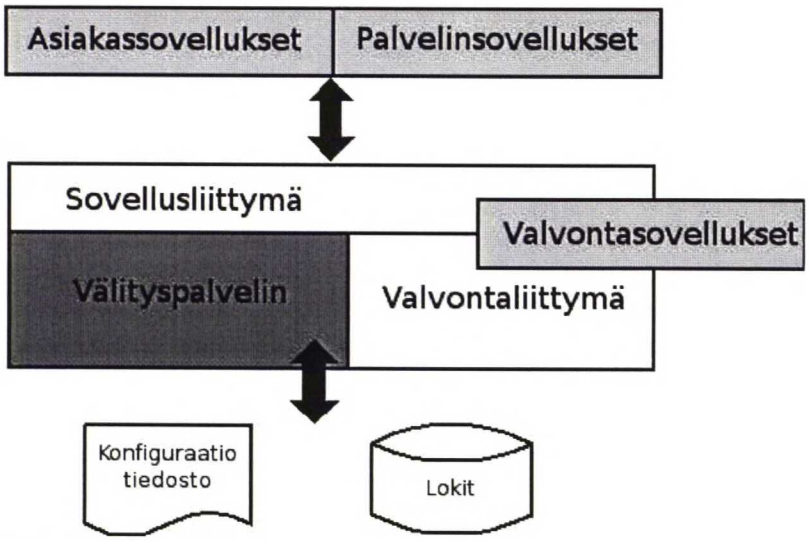
Palvelinsovellus voi myös itse lähettää viestejä, esimerkiksi jos asetukset tai käyttäjätiedot ovat toisen palvelinsovelluksen vastuulla eikä niitä löydy paikallisesta välimuistista, palvelinsovelluksen täytyy lähettää itse viesti näiden tietojen saamiseksi. Näin voidaan saadaan aikaan pitkiäkin ketjuja, joissa asiakassovelluksen pyyntö vaikuttaa useampaan palvelimeen. Kuorman jakamisen ja hajautuksen merkitys korostuu entisestään, koska mikäli välityspalvelin tukisi vain yhtä viestiä palvelinta kohti, yksinkertainenkin pyyntö voisi estää muiden yhtäaikaisten pyyntöjen käsittelyn.

Tärkeänä vaatimuksena esitettiin viestien yhteensopivuus eri asiakas- ja palvelinsovellusten välillä, segmenttirajapinta mahdollistaa tämän vaivattomasti. Jokainen viesti koostuu segmenteistä joita sovellusrajapinnalta pyydetään erikseen iteroiden, näin rajapintaa käyttävä sovellus ei edes näe niitä segmenttejä joista se ei tiedä. Samaten kun sovellukset käsittelevät viestien segmenttejä niitä kuvaavilla luokilla, eivät uudet kentät segmenttien lopussa näy vanhoille luokkien implementaatioille jotka eivät niistä tiedä. Niin kauan kuin viestissä ei muuteta segmenttien kenttien järjestystä, taikka vaihdeta segmenttejä viesteistä, ovat viestit aina versioiden välillä yhteensopivia. Saavutettu yhteensopivuus eri versioiden välillä on merkittävä etu, sillä nyt järjestelmää ei tarvitse koskaan päivittää kerralla. Järjestelmästä voidaan päivittää esimerkiksi vain muutaman käyttäjän asiakassovellus, jotka tarvitsevat uutta toiminnallisuutta.

Viestien selkokielisyys vaatii, että kaikkien viestien koodaus siirtotiellä on puhdasta tekstiä. Merkistöksi valittiin ISO-8859-1 [ISO8859] koodaus, jolla mahdollistetaan viestin helppo luettavuus kunhan se vain saadaan talteen verkkokaapparilla tai välityspalvelimen omalla jäljitystoiminnallisuudella. Järjestelmän lähiverkossa on muutamia työasemia hitaiden ISDN-yhteyksien takana [Tanen 02], joten siirtorajapinnan tuli tukea myös kompressiota, jolla näillä työasemilla ajettavat asiakassovellukset saatiin käytettäviksi. Sovellusrajapinta piilottaa tämän toiminnallisuuden sovellukselta käyttäen "deflate"-pakkausalgoritmia [RFC1951]. Pakatut viestit eivät ole enää helposti luettavissa, joten välityspalvelimen pitää purkaa pakatut viestit selkokieliseksi silloin kun jäljitys on kytketty päälle. Pakattuja viestejä on vaikeaa tulkita suoraan esimerkiksi verkkokaapparilla.

5 Suunnittelu

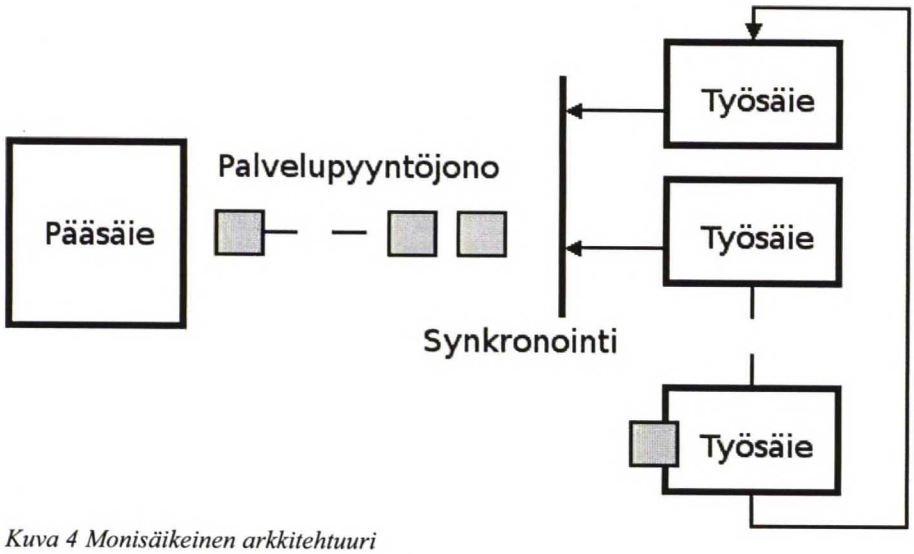
Kappale esittelee välitysohjelmiston ja sovellusrajapinnan suunnittelun sisältäen merkittävät suunnittelupäätökset ja syyt joiden vuoksi niihin päädyttiin. Välityspalvelimen suunnittelu jakautuu kolmeen eri osaan joista ensimmäinen ja tärkein on itse välityspalvelin. Toinen osa on sovellusrajapinta ja kolmas hallinta- sekä valvontakäyttöliittymä. Välityspalvelin on keskuskomponentti johon sekä sovellusrajapinnat että hallintakäyttöliittymät ovat yhteydessä.



Kuva 3 Komponentit

Suunnitteluun ryhdyttäessä oli jo päätetty käyttää TCP POSIX Socket [Posix 96] verkko-ohjelmointirajapintaa välityspalvelimen tietoliikenteessä. Ensimmäinen ratkaistava suunnitteluongelma oli transaktioiden rinnakkaisuus käyttäen valittua ohjelmointirajapintaa. Välityspalvelin on kokonaisuudessaan sovellustasolla [ISO7498] ja luottaa TCP-pinon tarjoamiin palveluihin. Tästä eteenpäin systeemikutsuja käsiteltäessä ei luettavuuden vuoksi viitettä ole merkitty erikseen, ellei toisin ole mainittu, käytetyt lähteet ovat [Stevens 92] sekä [Stevens 91].

Aluksi ilmeinen ratkaisu oli monisäikeisyyteen perustuvassa arkkitehtuurissa, jossa on helppoa antaa yksittäisten säikeiden käsitellä yksi transaktio kerrallaan. Tunnettu esimerkki arkkitehtuurista on Apache WWW-palvelin [Apache] . Palvelimessa yksi pääsäie vastaanottaa sisääntulevia yhteyksiä systeemikutsulla "accept" ja ohjaa ne sisäiseen jonoon. Palvelimessa on sisäinen säievaranto (eng. "threadpool"), jossa säikeet odottavat jonon semaforia [Tanen 93]. Pääsäikeen ohjattua uuden yhteyden jonoon se signaloi jonon semaforia , jolloin ensimmäinen odottava säievarannon säie saa yhteyden käsiteltäväkseen. Toteutukseen tarvitaan teoriassa vain yksi synkronoitu lukituspiste, eli jonon semafori.



Kuva 4 Monisäikeinen arkkitehtuuri

Käytännössä asiat eivät ole näin yksinkertaisia, suunnittelun edetessä kävi ilmeiseksi että tarvitaan suuri määrä synkronointipisteitä säikeille. Tarvittavia synkronointipisteitä olivat muun muassa tilastotietojen kerääminen, palvelinprosessien rekisteröinti, virhetilanteiden hallinta, hallintaliittymän palvelupyynnöt sekä lokitiedoston kirjoitus. Lopputuloksena on ohjelma joka on perusperiaatteeltaan yksinkertainen mutta jossa on huomattava määrä vaikeita synkronointiongelmia, joita ei ole yksikertaisesti mahdollista nähdä kaikkia ennalta. Tämä on totta erityisesti silloin kun ohjelmisto pyritään toteuttamaan siten, että alikomponentit pystyvät käyttämään suoraan toisiaan. Monisäikeisyyteen perustuvan välityspalvelimen arkkitehtuurin luonnosohjelma on esitetty listauksissa 1 ja 2. Verkkorajapinnan systeemikutsut on esitetty kursiivilla.

```
1. main start
2.    while (sc = accept()) != ERROR
3.    do
4.      enqueue(Q,sc)
5.    done
6. end main
```

Listaus 1 Monisäikeisen välityspalvelimen pääsäikeen luonnosohjelma

Rivillä 2 hyväksytään yhteys ja se sijoitetaan työjonoon, tämän jälkeen pääsäie palaa odottamaan uutta yhteyttä.

```
1. worker start
2.    while (sc = dequeue(Q)) != ERROR
3.    do
4.      req = readmsg(sc)
5.      sr = route_and_reserve(req)
```

```
6.      sendmsg(req,sr)
7.      resp = readmsg(sr)
8.      sendmsg(resp,sc)
9.      release(sr)
10.     close(sc)
11.   done
12. end worker
```

Listaus 2 Monisäikeisen välityspalvelimen työsäikeen luonnosohjelma

Rivillä kaksi poistetaan yhteys jonosta, riveillä 4-6 luetaan viesti ja lähetetään se oikealle palvelimelle. Riveillä 7-10 välitetään vastaus takaisin lähettäjällä, vapautetaan palvelin ja päätetään yhteys.

Välityspalvelimen ohjelmistoalusta HP-UX 10.20 tarjoaa "pthread"-rajapinnan [Posix 96] monisäikeisten ohjelmistojen luomiseen. Monisäikeinen välityspalvelimen arkkitehtuuri olisi ollut kiinnostava ratkaisu, sillä silloin yhtäaikaisten transaktioiden käsittely olisi voitu ohjelmoida rinnakkaisesti. Monisäikeisessä ohjelmoinnissa piilee kuitenkin edellä esitetty laadullinen ongelma. Tässä mallissa ohjelma on näennäisesti helppo suunnitella, koska tarvitaan vain yhden transaktion kerrallaan mahdollistava lineaarinen ohjelma, jota sitten monistetaan useille säikeille. Vaikka monisäikeinen arkkitehtuuri on helppo suunnitella ja esittää ymmärrettävästi, se on kuitenkin vaikea toteuttaa oikein ottaen huomioon kaikki rinnakkaisuuteen liittyvät ongelmat. Säikeiden käyttäminen ohjelmassa laajentaa mahdollisten tilojen määrää rajattomasti verrattuna yksisäikeiseen ohjelmaan, vikaantumistilanteessa ohjelmiston käyttäytymisen jäljittäminen ja virheen on erittäin vaikeaa, joskus jopa käytännössä mahdotonta.

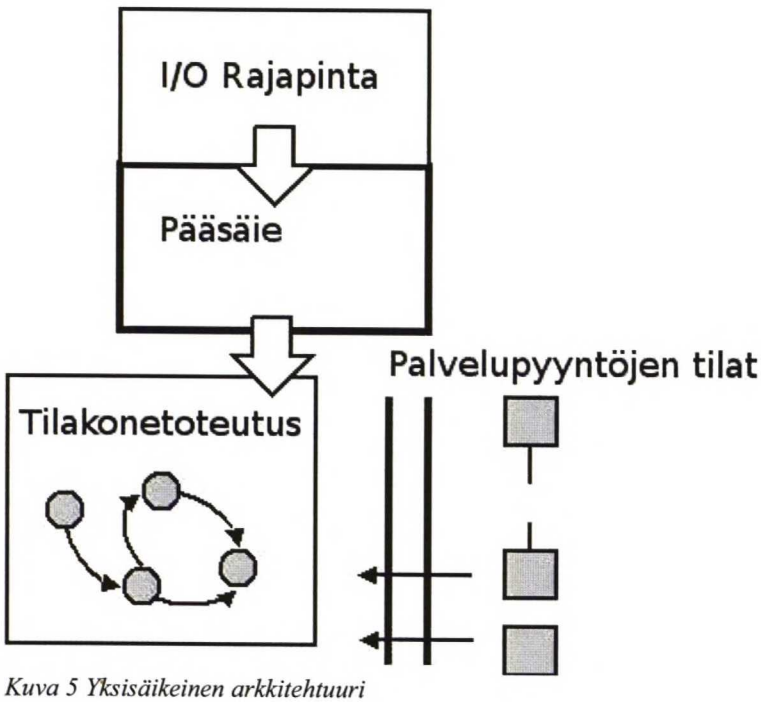
Monisäikeisyyteen perustuva arkkitehtuuri katsottiin liian vaikeaksi toteuttaa luotettavasti annetussa aikataulussa. Riskinä nähtiin myös se, että käyttöjärjestelmän säieympäristö ei olisi skaalautunut tarpeeksi ja sen aiheuttama ylimääräinen kuorma käyttöjärjestelmällä olisi voinut vaikuttaa välityspalvelimen suorituskykyyn.

Suunnittelussa päädyttiin toteuttamaan näennäisesti monimutkaisempi mutta selkeästi paremmin hallittava arkkitehtuuri käyttäen tila-suunnittelukuviota. (eng. "state-design pattern"). [Gamma 95] Välityspalvelimen välittämän viestiliikenteen eri perustilat on esitetty taulukossa 2.

Asiakassovellus	Palvelinsovellus
Passiivinen	Odottaa viestiä
Lähettää viestiä	Odottaa viestiä
Odottaa vastausta	Lukee viestiä
Odottaa vastausta	Prosessoi viestiä
Odottaa vastausta	Lähettää vastausta
Lukee vastausta	Odottaa viestiä
Prosessoi vastausta	Odottaa viestiä

Taulukko 2 Sovellusten tilat

Välityspalvelin täytyy suunnitella siten, että se pystyy käsittelemään kunkin perustilasiirtymän atomisesti ja nopeasti, näin yksisäikeisellä arkkitehtuurilla voidaan käsitellä suuri määrä transaktioita yhtäaikaisesti, vaikka todellisuudessa välityspalvelimessa tapahtuu vain yhden viestin yksi tilasiirtymä kerrallaan.



Kuva 5 Yksisäikeinen arkkitehtuuri

Tilakoneeseen perustuvaa arkkitehtuuria käytetään edelleen esimerkiksi WWW-palvelimissa koska tämän lähestymistavan etuna on resurssien säästäminen, sekä helppo kannettavuus eri alustoille säiemallista riippumatta. Arkkitehtuuri on tunnettu jo pitkään perinteisissä POSIX-ohjelmissa ja melko iso osa UNIX pohjaisten käyttöjärjestelmien omista prosesseista käyttää tätä arkkitehtuuria, kuten esimerkiksi NFS palvelin. [RFC1094].

Välityspalvelimen tilakonearkkitehtuuri perustuu lähteissä [Schm 00] ja [Gamma 95] esitettyihin malleihin. [Schm 00] esittää suunnittelun perusfilosofian, missä yksisäikeillä ohjelmistolla voidaan palvella pyyntöjä asynkronisesti, ja [Gamma 95] tilakone-suunnittelumallin. Välityspalvelimen tilakoneita ajetaan käyttöjärjestelmän antamilla tapahtumilla (eng. "event"). Sovellus näkee TCP-yhteydet kahvoina, joita kutsutaan vastakkeiksi (eng. "socket"). Käyttöjärjestelmä tarjoaa rajapinnan jolla voidaan rekisteröityä kuuntelemaan vastakkeiden tapahtumia. Näitä perustapahtumia on käytännössä vain 2.

- Vastake on kirjoituskelpoinen. Käyttöjärjestelmän vastakkeelle varaamassa lähetyspuskurissa on tilaa, mikä sallii tavujen lähettämisen vastakkeeseen siten, että lähettämiseen käytettävä systeemikutsu palaa välittömästi.
- Vastake on lukukelpoinen. Käyttöjärjestelmän vastakkeelle varaamassa vastaanottopuskurissa on valmiina tavuja, jolloin vastaanottamiseen käytettävä systeemikutsu palaa välittömästi.

Näillä ehdoilla voidaan viestiliikenne käsitellä yksisäikeisellä ohjelmistolla, sillä voidaan taata, että kaikki operaatiot kestävät "vähän" aikaa. Takeita käsittelyajasta ei tietenkään voida saada, mutta tässä tapauksessa sovelluksen täytyy vain luottaa käyttöjärjestelmään.

Käyttöjärjestelmä signaloi uudesta sisääntulevasta yhteydestä merkitsemällä palvelinvastakkeen (eng. "server socket") lukukelpoiseksi. Katkenneesta vastakkeesta signaloidaan samoin, mutta luku-

yritys tällaiseen vastakkeeseen palauttaa virhekoodin joka kertoo sovellukselle yhteyden katkeamisesta.

Vastaketapahtumien lisäksi tarvitaan joitain aikapohjaisia tapahtumia, näitä ovat esimerkiksi viestien aikarajojen tarkistamiseen liittyvät tilasiirtymät. Aikapohjaiset tapahtumat voidaan saada aikaan antamalla aikaraja vastiketapahtumien odotukselle käytettyyn systeemikutsulle. Aikaraja määritellään siten, että se laukeaa kun seuraava aikapohjainen tapahtuma tarvitaan jollekin tilakoneelle. Tällöin systeemikutsu palaa ilman vastaketapahtumia. On huomattavaa, että aikaraja tarkastetaan aina kun systeemikutsu palaa, koska tällöin käsitellään aina oikein tilanne, jossa aikarajan umpeutumishetkellä saadaan vastaketapahtumia. Aikarajaa verrataan joka käyttöjärjestelmän kellon aikaan ja jos tämä aika on sama tai myöhäisempi kuin aikaraja, suoritetaan tilakone aikatapahtumalla.

Yksisäikeisen välityspalvelimen luonnosohjelma on esitetty listauksessa 3.

```
1.  main start
2.  while running == TRUE
3.  do
4.   update_filesets(sm_list,timerl);
5.   timeout = next_timeout(timerl);
6.
7.   rval = select(readfs,writefs,timeout)
8.   if rval > 0
9.   then
10.     for each fd in readfs,writefs
11.     do
12.      if eventflag(fd)
13.      then
14.        execute(fd,get_sm(fd))
15.      end fi
16.    done
17.  end if
18.  for each t in timerl
19.   do
20.     if t < currenttime
21.     then
22.       execute(t,get_sm(t))
23.     end if
24.   done
25. done
26. end main
```

Listaus 3 Yksisäikeisen välityspalvelimen luonnosohjelma

Riveillä 4-5 valmistellaan "select"-systeemikutsua varten ne vastakkeiden kahvat, (eng. "file descriptor") joiden tapahtumista ollaan kiinnostuneita, sekä lasketaan seuraava aikaraja. Vastaketapahtumien odottaminen tapahtuu rivillä 7, minkä jälkeen tarkistetaan kuinka monta tapahtumaa saatiin ja ajetaan tilakoneita tapahtumilla. Aikapohjaiset tapahtumat ajetaan riveillä 18-24.

Välityspalvelimessa on yksi tilakone uusien pyyntöjen vastaanottoon, tämä tilakone myös luo uusia tilakoneita palvelemaan pyyntöjä ja rekisteröi ne välityspalvelimen sisäisiin tietorakenteisiin. Toinen aikatapahtumilla ajettava tilakone tarvitaan sisäiseen kirjanpitoon, tilastojen keruuseen ja lokin kirjoitukseen. Kolmas tilakone palvelee hallintakäyttöliittymän pyyntöjä ja palvelimien rekisteröintejä. Lopuksi on varsinainen viestin käsittelemiseen tehty tilakone. Kaikkien tilakoneiden toteutuksessa on paljon yhteistä, koska suurin osa toiminnoista perustuu viestien käsittelyyn. Viestin tyyppiä ei myöskään tiedetä ennen kuin se on kokonaan luettu, vasta tämän jälkeen tilakoneet erikoistuvat, näin olleen niiden alkupään tilasiirtymät ovat identtisiä. Viestien käsittelyyn tehdyt tilakoneet elävät niin kauan kuin yhteys asiakas- tai palvelinsovellukseen on olemassa, joten yksi tilakone voi käsitellä satoja viestejä elinkaarensa aikana.

Yksisäikeinen, tilakoneisiin perustuva arkkitehtuuri, on vaikeampi suunnitella ja vaatii huolellisempaa työtä kuin monisäikeinen arkkitehtuuri. Tilakoneiden etuina on kuitenkin helppo vian selvitys ja käytännössä deterministinen käyttäytyminen. Virhetilanteissa käyttöjärjestelmän tekemistä muistinvedoksista (eng. "core image") on suoraviivaisesti jäljitettävissä missä tilassa ohjelma oli kun käyttöjärjestelmä lopetti sen. Monisäikeisessä ohjelmistossa sama analyysi on huomattavasti vaativampaa jo pelkkien perkaimien (eng. "debugger") puutteiden takia, kun ne eivät hallitse kunnolla säikeitä. Samaten tilakoneisiin perustuvassa ohjelmistossa vika saadaan aina sopivalla syötteellä toistettua säännönmukaisesti. Monisäikeisessä ohjelmistossa pelkkä syöte ei riitä, vaan säikeiden ajojärjestys täytyy olla identtinen joka ajokerralla mikä ei tietenkään ole käytännössä mahdollista, koska vian sattuessa ei mistään voida tietää miten vikatilanteeseen saavuttiin. Yleistäen voidaan todeta, että yksisäikeisessä ohjelmistolla samalla syötteellä saadaan aina sama lopputulos, monisäikeisessä ohjelmistossa voi ympäristö, kuten käyttöjärjestelmän muut käyttäjät, vaikuttaa lopputulokseen merkittävästi.

Kolmantena vaihtoehtona harkittiin myös edellä esitettyjen arkkitehtuurien välimuotoa, jota käytetään vieläkin yleisesti joissain UNIX-järjestelmien palvelinprosesseissa. Kaupallisista sovelluksista esimerkiksi Oracle 9i-tietokanta käyttää tähän perustuvaa arkkitehtuuria. Moniprosessiarkkitehtuurissa on yksi pääprosessi , joka avaa kuuntelevan palveluvastakkeen ja odottaa yhteyksiä asiakassovelluksilta. Yhteyden saadessaan pääprosessi käyttää "fork" systeemikutsua joka aiheuttaa pääprosessin jakautumisen kahteen eri prosessiin joissa suoritus siten haarautuu. "fork"-systeemikutsun paluuarvo on erilainen isä- ja lapsiprosessille, joten suoritus voidaan haarauttaa koodissa tarkastelemalla tätä paluuarvoa. Vastikkeen kahva kopioituu kumpaankin prosessiin. Lapsiprosessi voi siten jatkaa palvelupyynnön suoritusta ja isäprosessi voi jatkaa yhteyksien vastaanottoa. Näin sekä isä- että lapsiprosessilla on samantyyppinen arkkitehtuuri, kuin ne olisivat säikeitä samassa prosessissa, mutta koska ne eivät jaa muistialueita, esimerkiksi yhteysongelmat yhdellä istunnolla eivät vaikuta muihin istuntoihin. Moniprosessiarkkitehtuurissa on kuitenkin suuria ongelmia, joiden vuoksi sitä ei kannata aina käyttää. Suurin ongelma on tiedon jako prosessien välillä, esimerkiksi kuinka varata ja vapauttaa palvelin viestin ajaksi ja raportoida mahdollisista ongelmista isäprosessille? Mainittu arkkitehtuuri sopii vain hyvin yksinkertaisille ja tilattomille järjestelmille joissa eri palvelevien prosessien ei tarvitse kommunikoida toistensa kanssa. Kommunikointia tarvittaessa päädytään helposti siihen, että pääprosessiin on toteutettava jonkinlainen tilakonejärjestelmä, jolloin ollaan lähtöpisteessä. Moneen prosessiin perustuva arkkitehtuuri on paikallaan jos palveluajat ovat pitkiä tai yksisäikeinen lähestymistapa on liian monimutkainen toteuttaa esim. kolmannen osapuolen kirjastojen takia.

Yksisäikeinen arkkitehtuuri on periaatteessa käyttäjän prosessiin rakennettu rinnakkaisuuden simu-

lointi. Pääsäie ajaa tilakoneita, eli ns. säikeitä, joissa tehtävän vaihto toiseen perustuu vapaaehtoisuuteen. Vapaaehtoisuuteen perustuva moniajo löytyy esimerkiksi Microsoft Windows 3.1 käyttöjärjestelmästä. Monisäikeinen arkkitehtuuri ratkaisee tehtävävaihdon käyttäjän puolesta, mutta ongelmana ovat resurssien lukitukset ja muut vastaavat tehtävät, jotka jäävät sovellusohjelmoijan vastuulle. Yksisäikeisen arkkitehtuurin suorituskyky on myös tehokkaampi, koska sen tarvitsee tehdä vain ne toimenpiteet tehtävien vaihtamiseksi, jotka ovat sille välttämättömiä. Monisäikeinen ohjelma hukkaa paljon suorituskykyään tehtävävaihtoon ja sen teho onkin paljon riippuvainen käytetystä käyttöjärjestelmästä, sekä siihen saatavista säie-toteutuksesta. HP-UX alustalla löytyy POSIX-thread kirjasto joka on standardi säikeiden toteuttamiseen C/C++-ohjelmointiympäristössä. Kyseinen rajapinta on vasta viime vuosina kypsynyt tarpeeksi luotettavaksi mutta välityspalvelinta suunniteltaessa se oli poissuljettu mahdollisuus.

Suorituskykyvaatimukset eivät käytännössä vaikuttaneet merkittävästi arkkitehtuurin valintaan, vaatimukset olivat niin kevyet verrattuna käytettävän alustan suorituskykyyn, ettei suorituskykyongelmaa millään oletuksella pääsisi syntymään. Pienikokoiset viestit myös suosivat yksisäikeistä tilakonearkkitehtuuria.

Reitittimen arkkitehtuuri määrittelee sovellusrajapintojen toiminnan eikä hallintakäyttöliittymä sinänsä vaikuta arkkitehtuuriin, mikäli se on tehty lähinnä ohjelmiston valvontaan. Hallintakäyttöliittymällä on vaikutusta arkkitehtuuriin, jos sillä on tarkoitus muuttaa järjestelmän konfiguraatiota ajon aikana tai muuten tehdä monimutkaisia operaatioita. Pelkkään valvontaan tehty hallintaliittymä voidaan tehdä yksinkertaisesti, koska toiminnot reitittimen ja hallintaohjelmiston välillä ovat rajoitettuja ja tilattomia.

Vaatimukset asettavat hallintaliittymälle muutamia reunaehtoja joista tärkein on viestien jäljitys sekä tilastotietojen keräys. Tilastojen keräys on yksinkertainen operaatio joka voidaan toteuttaa tavallisella viestillä, jolla pyydetään reitittimen keräämiä tilastotietoja. Viestien jäljitys on monimutkaisempaa toteuttaa. Päätettiin, että on varminta tehdä jäljitys siten, että välityspalvelin itse kirjoittaa tiedostoon järjestelmässä liikkuvat viestit jäljityksessä, hallintaliittymä vain kytkee tämän toiminnallisuuden päälle tai pois käyttäen esitettyjä suodatuskriteerejä Ratkaisu on huomattavasti luotettavampi ja helpompi toteuttaa kuin viestien välitys hallintakäyttöliittymään, koska transaktio-pohjainen viestien välitys ei tue vuomaista (eng. "streaming") tiedonvälitystä. Hallintaliittymän olisi pitänyt tehdä määräajoin kyselyjä välityspalvelimelle, johon olisi myös pitänyt rakentaa ylimääräistä toimintaa viestien puskuroimiseksi ja tallettamiseksi.

Päädyttiin ratkaisuun jossa hallintaliittymä on yksinkertainen Microsoft Windows-pohjainen ohjelma, joka kommunikoi sisäisillä viesteillä välitysohjelmiston kanssa. Viestien tukemat operaatiot rajoittuvat tilattomiin toimintoihin. Toiminnallisuuksiin sisältyy.

- Palvelimien viestimäärien ja viiveiden kysely.
- Asiakassovelluksien viestimäärien ja viiveiden kysely.
- Sisäisten jonojen tila ja ajoaika. (eng. "system uptime").
- Viestien jäljityksen asettaminen päälle ja pois.

Toteutuksen yhteydessä nämä ominaisuudet todettiin varsin riittäviksi ja käytännössä tuotannossa käytetään vain komentorivisovellusta joka käyttää samoja rajapintoja. Komentorivipohjainen-hallintaohjelma on helppoa integroida hallinta-skripteihin jotka ovat yksinkertaisia valikkoja tarjoavia sovelluksia. Ajamalla hallintaskriptejä etäpäätteeltä voidaan muun muassa kysellä palvelimien tiloja sekä käynnistää ja sammuttaa palvelimia.

Sovellusliittymät päätettiin tehdä rakenteella jossa prosessi rekisteröityy välityspalvelimeen käyttäen käyttöjärjestelmäkutsua joka tekee TCP-yhteyden välityspalvelimella. Yhteyden muodostettua liittymä lähettää palvelimelle rekisteröintiviestin. Onnistuneessa rekisteröinnissä liittymä saa positiivisen

vastauksen välityspalvelimelta ja muodostaa siitä istuntokahvan jonka se palauttaa sovellukselle.

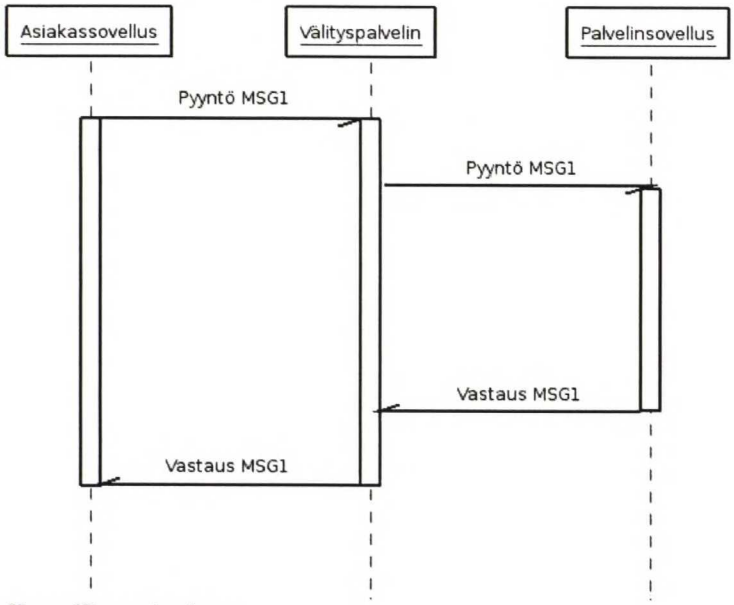
Asiakassovelluksen ja palvelinsovelluksen toiminnat poikkeavat rekisteröinnin jälkeen toisistaan. Asiakassovellusta ajaa pääasiassa käyttöliittymä tai jokin muu ulkoinen tekijä ja sovellus tekee viestejä vain reagoidessaan näihin tapahtumiin, esimerkiksi käyttäjän valitessa jonkin toiminnon.

Palvelinsovellus ajaa itseään silmukassa ja odottaa sovellusrajapinnalta viestejä, palvelee niitä ja lähettää asiakassovelluksille vastauksia samaa rajapintaa käyttäen. Palvelinsovellus toimii listauksessa 4 esitetyn luonnosohjelman mukaan. Sovellusrajapinnan funktiot on merkitty listauksessa kursiivilla.

```
1.  main start
2.    session = register(url,name);
3.
4.
5.    while reqmsg = receive(session);
6.    do
7.      respmsg = createempty(session,reqmsg);
8.      service_message(reqmsg,respmsg);
9.
10.     send(reqmsg,respmsg);
11.   done
12.   unregister(session);
13.end main
```

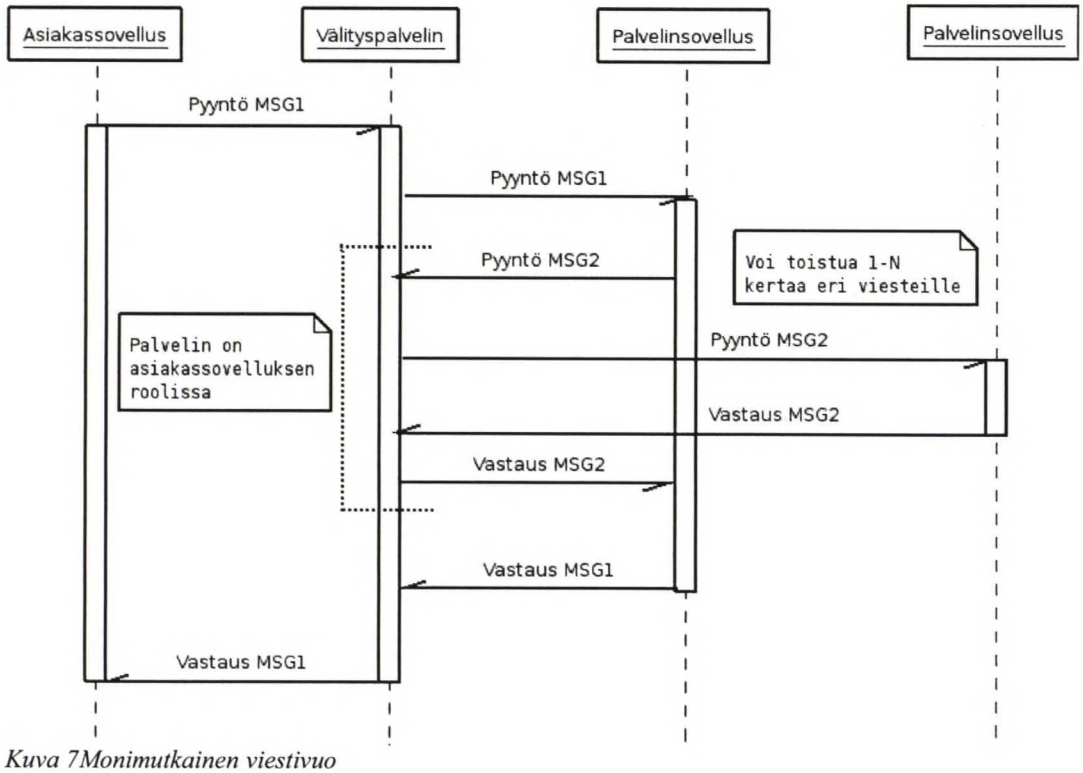
Listaus 4 Yksisäikeisen välityspalvelimen luonnosohjelma

Rivillä 7 luodaan tyhjä vastausviesti joka täydentää transaktion, mikäli palvelinsovellus luo viestin ilman pyyntöviestiä se katsotaan uudeksi viestiksi ja aloittaa uuden transaktion kun palvelinsovellus lähettää sen. Kuvassa 6 on esitetty perusviestivuo, jossa asiakassovellus kommunikoi palvelimen kanssa.



Kuva 6Perusviestivuo

Palvelinsovellus voi viestiä palvellessaan lähettää itsekin viestejä, jotka ovat riippumattomia viestistä jonka se vastaanotti. Palvelinsovelluksen täytyy pitää palvelupyyntöviestiä tallessa jotta sovellusrajapinta osaa sitoa vastausviestin oikeaan palvelupyyntöön. Kuvassa 7 on esitetty viestivuo tässä tilanteessa.



Kuva 7Monimutkainen viestivuo

Viestiprotokollan rakenne on yksinkertainen. Viesti koostuu segmenteistä jotka koostuvat kentistä. Kentät on erotettu puolipiste-merkillä ';' jonka ASCII-koodi on 0x3B [ISO646]. Mikäli kentässä on tämä merkki, se koodataan koodauksenvaihtomerkillä '\', ASCII-koodi 0x5C.

Viestin rakenteen BNF-esitys on esitetty listauksessa 5 [Naur 60].

```
1. <message> ::= <segment>
2.               { <segment> }
3.
4. <segment> ::= <field> ";" <field> { ";" <field> } ";"
5.
6.<field> ::= { <letter> | <digit> }
```

Listaus 5 Viestin BNF-esitys

Listauksessa "*digit*" tarkoittaa numeroa välillä 0-9 ja "*letter*" on mikä tahansa ISO-8859-1 aakkoston merkki. Viestille ei voida toteuttaa jäsennintä käyttämällä esimerkiksi jäsenningeneraattoria. Kieli jonka viesti kuvaa ei ole LARL(1)-yhteensopiva, jota jäsentävää koodia suurin osa jäsenningeneraattoreista tuottaa. Yhteensopimattomuus johtuu siitä, että ehdollisten kenttien lukumäärä määräytyy viestissä olevan arvon, eli literaalin perusteella. [Appel 97]

Segmentissä on aina vähintään kaksi kenttää, joissa ensimmäisestä on segmentin nimi ja toisessa on kenttien määrä segmentissä mukaan lukien mainitut pakolliset kaksi kenttää. Segmentti päättyy aina puolipistemerkkiin, sillä muuten ei olisi mitään tapaa kertoa milloin viesti päättyy, koska viesti-rakenteessa ei ole ilmoitettu tavumääriä.

Viestin semanttisessa rakenteessa ensimmäinen segmentti on aina otsakesegmentti nimeltään "CTL" joka määrittelee viestin nimen ja sen kuinka monta segmenttiä viestissä on. Tyypillinen viesti näyttää seuraavalta.

```
CTL;16;;21:31:22;control;@SMPS;;1;@CONTROL;N;N;0;Y;;;;@LIST;7;@ALL
SERVERS;;;;;
```

Viesti on esimerkissä nimeltään "@CONTROL" ja sen on lähettänyt sovellus nimeltään "control". Viestissä on yksi segmentti nimeltään "@LIST" ja se sisältää seitsemän kenttää joista vain yhtä on käytetty.

Koodinvaihtomerkkiä käytetään segmentissä erotinmerkkien suojaamiseen. Segmentissä tämä näyttää seuraavalta, jos segmentin kolmannessa kentässä on kirjaimellinen merkkijono "Hello;World".

```
TESTSEG;3;Hello\;World;
```

Merkkikoodaus vie resursseja, mutta sen tarvitsema aika on O(n) [Corm 96] ja se on laskennallisesti triviaali. Vaatimusten mukaan helposti luettava viestiliikenne on tärkeämpi ominaisuus kuin pitkälle optimoitu viestiliikenne.

Pakkaus linjalla on toteutettu siten, että kaikki otsakesegmenttiä seuraavat sovellussegmentit pakataan deflate-pakkauksella ja liitetään otsakesegmentin perään yhtenä suurena kenttänä. Merkkikoodaus on myös pakollinen koska pakattuun dataan voi tulla mitä tahansa merkkejä.

5.1 Käytettävyys ja Vikatilanteet

Palvelinsovellusten monimutkainen logiikka ja niiden suorittamat monimutkaiset operaatiot johtivat käytännössä siihen, että niihin tulikehitysvaiheessa lukuisia virheitä, joista osa aiheutti palvelimen vikaantumisen.

Välitysohjelman jäljitystoiminnon lisäksi kehitettiin myös muita käytettävyyttä parantavia ominaisuuksia. Vianselvityksen nopeuttamiseksi rakennettiin useita erilaisia menetelmiä joihin kuuluu aikaisemmin mainittu prosessien valvontaohjelmisto sekä lokitusjärjestelmä, joka kerää jatkuvasti palvelinprosessin koodiin kirjoitettuja lokiviestejä rinki-puskurimuistiin, johon mahtuu konfiguroitava määrä viimeisimpiä lokiviestejä. Palvelin saadessa signaalin joko käyttöjärjestelmältä tai prosessien valvontaohjelmistolta, se pystyy usein kirjoittamaan levylle viimeisimmät lokirivit joista pystyttiin päättelemään, mitä palvelin teki virhetilanteen sattuessa.

Prosessien valvontaohjelmisto pystyy signaloimaan prosessia, jos se päättelee, että palvelinsovellus on vikaantunut sekä lopettamaan sen. Valvontaohjelmisto voi käynnistää palvelinsovelluksen uudelleen, joten mikäli virhetilanne ei ole välittömästi toistuva, järjestelmään ei tule merkittävää palvelukatkosta. Testauksen ja tuotantoon oton yhteydessä havaittiin virheitä, joissa tietyntyyppinen viesti vikaannutti palvelinsovelluksen säännönmukaisesti. Esimerkiksi viestin puuttuva tai vääränmuotoinen parametri oli yleinen ongelman aiheuttaja, useimmiten nämä vikaantumiset olivat kuitenkin sattumanvaraisia ja vaativat tiettyä yhdistelmää sekä asiakassovelluksen että palvelinsovelluksen tilasta.

Palvelimen ripeä uudelleenkäynnistys auttaa piilottamaan vian käyttäjältä, mutta lisäksi välityspalvelimeen tehtiin vaatimuksen mukaisesti tuki usealle samaa viestiä kuuntelevalle palvelinsovellukselle. Toiminnallisuus toteutettiin määrittelemällä jokaiselle viestityypille jono johon liitetään lista aktiivisia palvelinprosesseja, jotka ovat mainostaneet kuuntelevansa kyseistä viestiä. Jonoihin merkitään jokaista palvelinta kohti kuinka monta tätä nimenomaista viestiä on jonottamassa palvelimeen ja uuden pyynnön tullessa sisään valitaan aina se palvelin johon on lyhyin jono.

Lajittelua ei tehdä enää uudestaan siinä tapauksessa jos jokin palvelinprosessi alkaa toimimaan odottamattomasti nopeammin kuin muut ja tyhjentää oman jononsa hyvin nopeasti. Lajittelu tehdään uudestaan vain jos jokin palvelinprosessi menetetään. Jonotuksessa on mahdollista, että viestiä ei ehditä palvella aikarajan sisällä mikäli viesti on vielä jonossa joten sille ei käytännössä ole mitään jonotusaikarajaa, ainoastaan käsiteltävänä olevalla viestillä on aikaraja.

Palvelimen menetys voidaan havainnoida käytännössä vain kahdella erilaisella tavalla.

- Palvelimen vastakkeeseen tehtävä operaatio epäonnistuu, eli kirjoitus tai luku-operaatio vastakkeeseen palauttaa virhekoodin. Tämä virhetilanne tulkitaan siten, että palvelinprosessi lopetettu jolloin käyttöjärjestelmä on sulkenut yhteydet.
- Palvelimesta odotettava palvelupyynnön vastaus ei saavu annetun aikarajan sisällä. Tämä virhetilanne voi olla seuraus siitä, että sovelluspalvelin on joko liian kuormitettu tai sitten palvelinprosessi on vikaantunut.

Jälkimmäinen tilanne on ongelmallinen koska palvelinprosessi voi olla täysin toiminnallinen, mutta juuri tällä hetkellä se ei pysty vastaamaan. Välityspalvelin ei mitenkään tietää mistä on kyse joten sen pitää olettaa pahinta ja sulkea yhteys palvelimeen.

Yhdessä palvelinprosessia voi olla kerrallaan vain yksi palvelupyyntö kerrallaan käsittelyssä, muiden odottaessa välityspalvelimen sisäisessä jonossa. Virhetilanne käsitteellään palauttamalla virheviesti käyttäjälle, joka oli lähettänyt alkuperäisen viestin. Viestiä ei uudelleen lähetetä toiselle palvelinsovellukselle, koska välityspalvelin ei kaikissa tapauksissa tietää onko viesti käsitelty palvelimes-

sa vai ei. Viestin lähettäminen uudelleen palvelimelle voisi johtaa odottamattomaan tilaan. Viestin kirjoituksen epäonnistuttua palvelimelle on mahdollista sanoa, että palvelin ei saanut viestiä vastaan, jolloin viesti voitaisiin lähettää uudelleen, tätä ei kuitenkaan kannata tehdä koska se tekisi tilakoneesta turhan monimutkaisen vain yhtä harvinaista erikoistapausta varten.

Sovellusten valvontaohjelmisto tekee tuotannossa jatkuvia kevyitä kyselyjä, joka myös vähentää todennäköisyyttä, että asiakassovelluksen viestin lähettäminen olisi tilanne, jossa palvelimen vikatilanne havainnoitaisiin. Valvontaohjelmisto voi lopettaa ja uudelleen käynnistää palvelimen. Valvontaohjelmiston lähettämät kyselyviestit poistavat mahdollisen ongelman joka ilmenee silloin kun palvelinprosessi käynnistetään uudelleen ja välityspalvelin ei ole vielä huomannut, että palvelinprosessi on menetetty. Tässä tilanteessa palvelinprosessin rekisteröinti epäonnistuu, koska välityspalvelin luulee, että samanniminen palvelin on jo käytössä. Valvontaohjelmisto lähettää pyynnön reitittimelle palvelinsovelluksen poistamiseksi rekisteristä ennen kuin se yrittää palvelimen uudelleenkäynnistystä. Esitetty välityspalvelimen käyttämä N-1 varamalli on käsitelty lähemmin teoksessa [Marcus 00].

Asiakassovelluksia vasten tapahtuvat ongelmatilanteet eivät käytännössä vaadi mitään erikoiskäsittelyä, koska ongelmat ilmenevät tilanteissa jotka ovat reitittimen näkökulmasta tilattomia; viestin luku tai vastauksen kirjoitus. Palvelinprosessi ei myöskään piittaa siitä meneekö viesti asiakassovellukselle perille vai ei, viestit on tehty mahdollisimman tilattomiksi. Käytännössä välityspalvelin katkaisee yhteyden asiakassovellukseen jos se havaitsee virheen yhteydessä.

5.2 Suorituskyky ja Rajoitukset

Viestiliikenne on toteutettu siten, että välityspalvelin lukee jokaisen viestin kokonaisuudessaan, sekä pyynnön että vastauksen, ennen kuin se välitetään eteenpäin. Käytäntö vaatii enimmäisarvion välityspalvelimen muistivaatimuksista. Välityspalvelinprosessi tarvitsee vakiomäärän muistia omaan toimintaansa ja jos huomioidaan vielä palvelimien rekisteröitymiseen ja vastaavaan kuluva enimmäismuistimäärä päästää melko tarkkaan arvioon. Välityspalvelimen käyttämä staattinen enimmäismuistimäärä on edellä mainittu vakiomäärä, sekä lisäksi lasketaan suurin rinnakkaisten transaktioiden määrä kertaa suurin mahdollinen viestin koko.

Vaatimusten sadan yhtäaikaisen transaktion määrä on vakio, joten viestin enimmäiskoko voidaan määritellä. Enimmäiskooksi valittiin 4 megatavuksi, joten suurin välityspalvelimen vaatima muistimäärä on 400 megatavua. Käytännössä tämä on enemmän kuin palvelinkone pystyy tarjoamaan välityspalvelimelle mutta käytännössä tämä ei ole ongelma. Keskimääräinen järjestelmässä välitetty viesti on merkittävästi pienempi, käytännön maksiviestit liikkuvat 50-2000 kilotavun välillä keskimääräisen viestin ollessa muutaman kilotavun kokoinen. Välitysohjelmiston odotettavissa oleva enimmäismuistintarve on siten 5-10 megatavua. Viestien enimmäiskokoa ei voitu pienentää, koska muutama harva raportoinnissa käytetty viesti saattoi olla muutaman megatavun kokoinen. Raportointiviestiä käytettiin vain harvoin ja silloinkin vain yöllä tausta-ajona, kun järjestelmässä oli muutoin vain vähän käyttöä.

Päätettiin, että välityspalvelimeen ei tehdä mitään muuta resursseja rajoittavaa erikoiskäsittelyä kuin viestien enimmäiskoon ja transaktioiden määrän rajoitus. Käytännössä ei ollut järkevää tehdä monimutkaista resurssien valvontaa, jossa laskettaisiin kulloinkin käsittelyssä olevien viestien koko ja arvioitaisiin välityspalvelimen kuluttamaa muistimäärää. Riskinä tässä lähestymistavassa on se, että välityspalvelin voidaan ajaa triviaalisti tilaan jossa käyttöjärjestelmä ei pysty tarjoamaan sille tarpeeksi muistia. HP-UX käyttöjärjestelmässä on myös se ominaisuus, että se lopettaa liikaa muistia kuluttavan prosessin jos sen heittovaihtotiedosto (eng. "swap file") uhkaa täyttyä. HP-UX varmistaa tällä suunnittelufilosofialla, että pääkäyttäjä (eng. "root") pääsee koneeseen korjaamaan ongelman tilanteessa jossa hallinnasta karannut prosessi vie liikaa resursseja mikä normaalisti ehkäisisi esi-

merkiksi etäterminaalipalvelun käynnistämisen. Välityspalvelimen vikaantuminen muistin loppuessa on täysin mahdollinen skenaario, se on kuitenkin tuotannossa hyvin epätodennäköinen ilman muita suuria ongelmia, jotka aiheuttavat järjestelmän vikaantumisen. Järjestelmän vikaantuessa näin pahasti koko järjestelmä käynnistettäisiin muutenkin uudestaan, joten välitysohjelmiston virheenkäsittelyllä ei olisi mitään virkaa. Reitittimessä ei siis ole periaatteessa riittävää resurssien hallintaa, mutta tämä katsottiin hyväksyttäväksi.

Välityspalvelimen arkkitehtuurissa yksisäikeinen pääsilmukka käy tilakoneita läpi aina vastake-tapahtumia saadessaan. Tilakonemalli vaatii, että kaikki tilakoneiden operaatiot ovat hyvin nopeita eikä ole riskiä, että ne jäisivät pitkäksi aikaa suoritukseen. Nopea tässä yhteydessä on suhteellinen käsite, mutta sen voi katsoa tarkoittavan tapahtumaa joka ei kestä yli kymmentä millisekuntia jolloin päästään 100Hz suoritustaajuuteen.

Nopeusvaatimus rajoittaa tilakoneen tilafunktiossa suoritettavien primitiivioperaatioiden määrää. Aluksi monimutkainen tilafunktio voidaan jakaa pienempiin osiin kunnes tilafunktiot ovat tarpeeksi pieniä ollakseen ennustettavia. Erityisesti viestin lukua ja kirjoitusta varten asetetaan enimmäispuskurikoko, joten pystytään aina varmasti sanomaan, että luku- ja kirjoitusoperaatiot vastakkeisiin kestävät aina vakioajan.

Tilakoneet voidaan sijoittaa hajautustaulukkoon, jossa hakuavaimena käytetään vastakkeiden kahvoja. Tapahtuman saadessaan pääsilmukan toteutus etsii taulukosta tilakoneen ja kutsuu sen tilafunktiota. Vastakkeen kahva on hyväksyttävä hajautustaulukkoavain sellaisenaan koska käyttöjärjestelmä takaa sen ainutlaatuisen arvon. Suunnittelussa päädyttiin kuitenkin paljon yksinkertaisempaan ratkaisuun, koska enimmäistransaktioiden määrä on rajoitettu sataan transaktioon on myös aktiivisten tilakoneiden määrä enimmäismäärä 100, lisäksi tulee muutama välityspalvelimen sisäinen tilakone. Näin pienen tilakonemäärän läpikäynti ei tarvitse mitään erikoista hakutaulukkoja joten tilakoneet voidaan sijoittaa listaan, jonka läpikäynti onnistuu aina vakioajassa O(1) [Corm 96]. Arkkitehtuuripäätöksen vaikutti myös se, että käytetty "select" systeemikutsu haluaa vastakkeiden kahvat taulukossa, jonka sovellus joutuu käymään aina läpi lineaarisesti, tämänkään takia ei olisi kannattanut käyttää suoraa indeksointia.

Aikatapahtumia odottavat tilakoneet sijoitetaan omaan listaansa, jonka koko ei kasvaa koskaan suuremmaksi kuin yhtäaikaisten transaktioiden määrä. Lista käydään läpi joka kerta kun pääsilmukka suoritetaan ja tilakoneen tilafunktion suoritus tehdään mikäli nykyinen aika on sama tai ohi listaan merkitystä. Kun löydetään ensimmäinen tilakone, joka odottaa tulevaisuudessa tapahtuvaa tapahtumaa voidaan listan läpikäynti lopettaa. Tämä on mahdollista kun aikatapahtumia odottavat tilakoneet ovat listassa aikajärjestyksessä.

Prosessoritehoa kasvattamalla voidaan päästä nopeampaan välitysohjelmiston toimintaan, mutta yksisäikeisen ohjelman suorituskykyä rajoittaa yhden prosessorin teho, koska käyttöjärjestelmä ei voi jakaa sen kuormaa monelle prosessorille. Monisäikeinen ohjelma skaalautuisi paremmin, kun eri säikeet voisivat olla suorituksessa eri prosessoreilla. Tämä on kuitenkin ongelma, jota ei voi valitulla arkkitehtuurilla mitenkään ratkaista ja yhden prosessorin teho (400Mhz PA-RISC) oletettiin riittäväksi.

Viivevaatimukset eivät kuvatulla ratkaisulla ole kovinkaan relevantteja, koska järjestelmän rakenteessa ei juurikaan ole "ilmaa". Ainoa asia mitä voitaisiin selkeästi nopeuttaa on viestin rakenne, jossa viestin pituus olisi aina yksiselitteisesti protokollassa mukana, eikä välityspalvelimen tarvitsisi käydä koko viestiä läpi sen analysoimiseksi. Tämä kuitenkin johtaisi viestirakenteeseen josta helposti tulisi vaikeasti luettava.

Edellä mainituilla perusteilla voidaan päätellä, että suorituskyvyn kannalta listat ovat riittävä tietorakenne.

5.2.1 Sovellusten hajautus

Sovellusten hajautus on reitittimen näkökulmasta triviaalia, koska alun perin niiden sijaintia ei ole mitenkään arkkitehtuurissa rajoitettu ja palvelimet ovat vastuussa yhteyksien luomisesta. Välityspalvelimen ei tarvitse tietää missä palvelimet sijaitsevat. Yhteydet välityspalvelimen ja sovellusten välillä ovat standardeja TCP-yhteyksiä, joiden muodostaminen käytetyllä ohjelmointirajapinnalla on riippumaton siitä ovatko prosessit samassa vai eri palvelinkoneessa.

Arkkitehtuuri sallii hajautuksen jossa jokainen asiakas- ja palvelinsovellus on omassa fyysisessä koneessa, mutta käytännössä tämä ei ole käyttökelpoinen ratkaisu. Huomattavasti merkittävämpi etu on se, että vanhoihin järjestelmiin voidaan rakentaa yhteensopivaa viestirajapintaa käyttäen sovellusrajapintaa. Uudet asiakassovellukset voivat käyttää helposti vanhaa järjestelmää ilman, että niiden täytyy tietää mitään vanhasta järjestelmästä. Asiakassovellusten konfigurointi on myös erittäin helppoa koska ainoa konfiguraatio mitä järjestelmänhallitsijan tarvitsee tehdä on luoda tunnus ja salasana käyttäjälle, sekä asettaa sille välityspalvelimen URL. Palvelimen puolella tarvitaan pelkästään välityspalvelimen URL, mitään muuta konfiguraatiota ei välityspalvelimen suhteen tarvita.

Verkon kaistanleveys ei ole merkittävä tekijä järjestelmä käytölle, koska liikennettä asiakassovellusten ja palvelinsovellusten välillä on vain silloin kun käyttäjä suorittaa jotain tapahtumaa. Näin viestien väli on suhteellisen harva per käyttäjä, ja koska viestit on pitkälle optimoituja ne kuluttava vain vähän verkkoresursseja. Tuotannossa havaittiin, että järjestelmässä aktiivinen käyttäjä teki keskimäärin vain 2 kertaa minuutissa jotain sellaista, josta generoitiin viesti.

Asiakassovelluksen sijaitessa modeemi- tai ISDN-linjan takana voidaan käyttää viestiprotokollan tukemaa kompressiota. Viestien formaatti, puolipisteellä erotellut kentät, kompressoituvat erittäin hyvin joten pakettikoko saadaan merkittävästi pienemmäksi mikä nopeuttaa käyttäjäkokemusta hitailla linjoilla. Kompressiolla voidaan siten mahdollistaa hyvä käyttökokemus hyvinkin hitaan verkkolinkin yli.

Palvelimien hajautuksessa käytännössä ongelmana ovatkin pääasiassa välityspalvelimesta riippumattomat resurssit jotka saattavat olla laiteriippuvaisia. Mikäli kaksi palvelinta tarvitsee pääsyn samaan tietokantaan, ne joudutaan käytännössä aina asentamaan samaan koneeseen. Palvelimet jotka kirjottavat raportteja tai vastaavia lokeja joudutaan myös pitämään aina samassa koneessa koska ne olettavat tiedostojen löytyvän yhdestä paikasta.

5.3 Tilakoneet

Tilakonemalliin perustuva arkkitehtuuri sitoo ohjelmiston tiukasti pääsilmukan ympärille, eikä tilakonemallista voi paljoakaan poiketa, vaikka jossain tilanteessa muunlainen arkkitehtuuri olisi käytännöllisin. Välityspalvelimen sisäiset toiminnot, kuten tilastokirjoitus sekä hallintakäyttöliittymän palvelurajapinta, joudutaan myös tekemään tilakoneilla. Välityspalvelin on luontevinta toteuttaa yhtenä prosessina, koska muuta rinnakkaista käsittelyä ei tarvita. Tässä yhteydessä täytyy kuitenkin mainita välityspalvelimen lisäksi toteutettu prosessinhallintaohjelmisto, joka oli alun perin tarkoitus toteuttaa välityspalvelimen yhdeksi tilakoneeksi. Käytännössä päätettiin toteuttaa tämä ohjelmisto omana prosessinaan, eli kyseessä oli yhden välityspalvelimen toiminnallisuuden ulkoistaminen omaan prosessiinsa. Ratkaisu mahdollisti itse välityspalvelinprosessin valvomisen riippumattomalla järjestelmällä.

Asiakassovelluksen lähettämä viesti muodostaa transaktion, eli lähetettyyn viestiin on aina saatava synkronisesti vastausviesti. Näitä kutsutaan pyyntöviestiksi ja vastausviestiksi jotka muodostavat yhdessä transaktion täysin samalla tavalla kuin HTTP protokollassa tehdään. [RFC2616] Transaktio malli rajoittaa yhden sovelluksen tekemään yhden transaktion kerrallaan, sekä asiakas- että palvelinpuolella. Pyyntö- ja vastausviesti eivät poikkea rakenteeltaan toisistaan.

Listauksessa 3 esitetty välityspalvelimen pääsilmukan toteutus on hyvin yksinkertainen, hoitaen vain tilakoneiden tilafunktion kutsumisen tapahtumien perusteella. Järjestelmän varsinainen toiminnallisuus on tilakoneiden tilafunktiossa. Tilakoneet on toteutettu periaatteella, jossa ne ensin lukevat viestin kokonaisuudessaan sisään, kelpuuttavat ja lopuksi tarkistavat mikä viestin nimi on. Mikäli viesti on osoitettu itse välityspalvelimelle, kuten tilastopyyntö tai palvelimen-rekisteröintiviesti, tilakone rakentaa vastausviestin ja siirtyy tilaan jossa viesti kirjoitetaan takaisin viestin lähettäjälle. Normaalit palvelinviestit hoidetaan varaamalla palvelin viestin käyttöön ja siirtymällä tilaan jossa viesti kirjoitetaan palvelimelle. Vastausviestin saavuttua palvelimelta palvelimen varaus poistetaan ja siirrytään tilaan jossa viestiä kirjoitetaan asiakassovellukselle. Tässä tilassa ei ole mitään väliprosessointia, vaan viesti lähetetään tarkastamatta lähettäjälle.

Tilakoneet jaetaan palveluviestin käsittelyyn, eli normaalien järjestelmän viestin välittämiseen, sekä hallintaviesteihin jotka on tarkoitettu itse välityspalvelimelle. Taulukossa 3 on esitetty palveluviestin tilakone ja taulukossa 4 on hallintaviestin tilakone. Tilakoneet on esitetty käyttäen dokumentista [WTP] lainattua notaatiota, jossa kullakin rivillä on kolumnit tilakoneen tilaa, tapahtumaa, ehtoa, seuraavaa tilaa ja odotettavaa tapahtumaa varten. Taulukkoa luetaan seuraavasti: jokaisella rivillä kolumnissa *tila* on tilakoneen tila. Kolumni *tapahtuma* kuvaa tapahtumaa, joka tilakoneelle voidaan tässä tilassa antaa, mitkään muut tapahtumat eivät muuta tilakoneen tilaa. Tapahtuma on se, joka aiheuttaa aiheuttaa tilasiirtymän tai vaihtoehtoisesti paluun samaan tilaan. Kolumnissa *ehto* on ehto, jonka pitää täyttyä, että voidaan jatkaa seuraavaan tilaan. Kolumnissa *seuraava tila* kerrotaan tila, johon siirrytään jos annettu ehto on täyttynyt. Lopuksi *odotettu tila* kolumni kuvaa sitä, mitä vastake- tai aikapohjaistapahtumaa tilassa odotetaan seuraavaksi. Odotettu tila on tärkeä välityspalvelimen toteutukselle, koska sillä voidaan minimoida parametrien määrä "select" systeemikutsulle, mikä parantaa suorituskykyä ja toteutuksen laatua.

Reitittimessä on myös muita yksinkertaisempia ja erikoistuneita järjestelmänhuoltotilakoneita (eng. "house keeping"), jotka odottavat ajastetun tapahtuman laukeamista. Näitä tilakoneita ovat esimerkiksi lokikirjoitukset ja vastaavat operaatiot, jotka ovat käytännössä vain pääsäikeen säännöllisin väliajoin suorittamia toimintoja. Eräs toinen järjestelmänhuoltotilakone on menetelmä, jolla uusia palvelupyyntöjä otetaan vastaan. Systeemikutsu "select" palauttaa kuunnellulle palvelinvastakkeelle tapahtuman merkiten sen lukukelpoiseksi. Pääsäie tarkistaa kerran onko yhteyteen tulossa uusi yhteydenottopyyntö. Jos näin on, pääsäie voi hyväksyä uuden yhteyden (ellei samanaikaisia transaktioita ole jo liikaa) ja luoda uuden tilakoneen transaktiota palvelemaan. Yhteyttä hyväksyttäessä tarkistetaan myös onko sovelluksen IP-osoite sallitulla listalla, tällä voidaan hieman kontrolloida sitä mitkä työasemat saavat ottaa yhteyden järjestelmään. Käyttäjien tunnistus ja valtuutus eivät ole välityspalvelimen vastuulla, vaan tunnukset ja salasanat kulkevat palveluviesteissä mukana.

Tila	Tapahtuma	Ehto	Seuraava Tila	Odotettu tapahtuma
INITIAL	-		CLIENT_READ	CS_READ_RDY
CLIENT_READ	CS_READ_RDY	NOT_READ	CLIENT_READ	CS_READ_RDY
		READ	WAIT_SRV	WAIT_SERVER
		ERROR	TERMINATE	
	TIMEOUT		TERMINATE	
WAIT_SRV	TIMEOUT		CLIENT_WRITE	CS_WRITE_RDY
	WAIT_SERVER	RESERVED	SRV_WRITE	SS_WRITE_RDY
		RESERVE_FAULT	CLIENT_WRITE	CS_WRITE_RDY
SRV_WRITE	SS_WRITE_RDY	NOT_WRITTEN	SRV_WRITE	SS_WRITE_RDY
		WRITTEN	SRV_READ	SS_READ_RDY
		ERROR	CLIENT_WRITE	CS_WRITE_RDY
	TIMEOUT	ERROR	CLIENT_WRITE	CS_WRITE_RDY
SRV_READ	SS_READ_RDY	READ	SRV_READ	SS_READ_RDY
		NOT READ	CLIENT_WRITE	CS_WRITE_RDY
		ERROR	CLIENT_WRITE	CS_WRITE_RDY
	TIMEOUT	ERROR	CLIENT_WRITE	CS_WRITE_RDY
CLIENT_WRITE	CS_WRITE_RDY	NOT WRITTEN	CLIENT_WRITE	CS_WRITE_RDY
		WRITTEN	CLEANUP	
		ERROR	CLEANUP	
	TIMEOUT	ERROR	CLEANUP	
CLEANUP			TERMINATE	

Taulukko 3Palveluviestin tilakone

Hallintaviestin käsittelyyn tehty tilakone on merkittävästi yksinkertaisempi kuin palveluviestitilakone, sillä sen ei tarvitse lähettää viestiä mihinkään vaan yksinkertaisesti koostaa vastausviesti ja lähettää sen saman tien takaisin. Hallintaviestitilakoneen ei tarvitse tarkistaa aikarajoja tai palvelimen terveyttä.

Tila	Tapahtuma	Ehto	Seuraava Tila	Odotettu tapahtuma
INITIAL	-		CLIENT_READ	CS_READ_RDY
CLIENT_READ	CS_READ_RDY	NOT READ	CLIENT_READ	CS_READ_RDY
		READ	CLIENT_WRITE	WAIT_SERVER
		ERROR	TERMINATE	
	TIMEOUT		TERMINATE	
CLIENT_WRITE	CS_WRITE_RDY	NOT_WRITTEN	CLIENT_WRITE	CS_WRITE_RDY
		WRITTEN	CLEANUP	
		ERROR	CLEANUP	
	TIMEOUT	ERROR	CLEANUP	
CLEANUP			TERMINATE	

Taulukko 4 Hallintaviestin tilakone

Taulukossa 5 on esitetty eri tilojen merkitykset tilakoneessa. Tilakoneet reitittimen sisällä koostuvat kaikki samoista perustiloista. On huomattava, että vaikka kyseessä ovatkin sinänsä formaalit tilat, on toteutuksessa joitain poikkeuksia. Poikkeukset voivat muuttaa hieman, sitä millä ehdoilla tilasiirtymät tapahtuvat. Täysin formaaliin esitykseen pitäisi sisältää myös nämä poikkeukset, eli jokainen mahdollinen kombinaatio. Tämä ei käytännössä ole tarkoituksenmukaista.

Monet konfiguroitavat parametrit vaikuttavat siihen kuinka tilakone toimii vaikka sen syöte, eli tapahtumat, olisivat samoja. Esimerkiksi, jos viestien jäljitys on asetettu päälle, täytyy kaikkien lukutilojen päätteeksi kirjoittaa levylle viestin sisältö. Tällainen operaatio joka sisältää kirjoitusta levylle pitäisi ottaa osaksi tilakonetta, koska on riski ettei kirjoitus onnistu ja käyttöjärjestelmä asettaa välityspalvelimen prosessin odottamaan. Toteutuksessa luotetaan kuitenkin siihen, että tässä tapauksessa levylle kirjoitus ei koskaan epäonnistu siten, että "write" systeemikutsu jää odottamaan (eng. "I/O wait"). Tilanne on kuitenkin täysin mahdollinen esimerkiksi silloin, kun kirjoitettava levy on NFS [RFC1094] partitio ja käyttöjärjestelmä on konfiguroitu siten, että menetettäessään yhteyden verkkolevyyn käyttöjärjestelmä yrittää operaatiota uudestaan kunnes se onnistuu. Kuvatussa tilanteessa välityspalvelin voisi olla pysähdyksissä useita minuutteja.

Toinen samantyyppinen poikkeama tilakoneesta liittyy palvelimen jonotukseen. Mikäli on konfiguroitu, että palvelinta ei jäädä odottamaan jos jono sinne on pidempi kuin N-transaktiota, ei tilakone mene koskaan tilaan "WAIT_SERVER". Tilakone siirtyy suoraan tilaan "CLIENT_WRITE", jossa asiakassovellukselle kirjoitetaan virhevastausta. Tiukasti ajatellen pitäisi tehdä erilainen tila jokaiselle jonon pituuden arvolle, mutta tämä ei ole käytännöllistä. Sama koskee myös tilakoneen käyttämiä luku- ja kirjoituspuskureita, joista riippui kuinka monta vastaketapahtumaa tarvitaan että koko viesti on luettu sisään ja voidaan siirtyä seuraavaan tilaan.

Tila	Merkitys	Huomioitavaa
INITIAL	Alkutila josta tilakone käynnistetään. Tästä vaiheesta siirrytään tilaan jossa luetaan viestiä sisään.	
CLIENT_READ	Viestiä luetaan kunnes koko viesti on luettu tai sitten sen lukeminen epäonnistuu.	Seuraava tilasiirtymää määräytyy sen mukaan onko kyseessä tavallisen palveluviestin tai hallintaviestin käsittely. Tämä tiedetään vasta kun koko pyyntö on luettu.
WAIT_SRV	Tilakone odottaa kunnes sen tarvitsema palvelin on varattu tämän transaktion käyttöön.	Tämä tila voi päättyä myös virheeseen jos palvelinta ei ole saatavilla tai jonotus on kestänyt säädetyn aikarajan yli.
SRV_WRITE	viestiä kirjoitetaan varatulle palvelimelle.	Palvelin poistetaan rekisteristä jos tämä tila saa virhetapahtuman.
SRV_READ	Vastausta luetaan palvelimelta.	Kun tästä tilasta siirrytään kirjoittamaan vastausta lähettäjälle vapautetaan ensin palvelin. Palvelin poistetaan rekisteristä jos tämä tila saa virhetapahtuman.
CLIENT_WRITE	Vastausta kirjoitetaan lähettäjälle.	Tähän tilaan voidaan hypätä suoraan monesta muusta tilasta, koska tässä voidaan kirjoittaa virheilmoitusta asiakassovellukselle jos käsittely on epäonnistunut jossain muualla.
CLEANUP	Siivoustila jossa kerätään tilastot ja mahdollisesti kirjoitetaan tietoja lokille.	Alkutilaan siirrytään vasta kun uusi pyyntö on saatu hyväksyttyä.

Taulukko 5 Tilat

Taulukossa 6 on esitetty tapahtumat, jotka siirtävät tilakoneita eteenpäin. Useimmiten tapahtuma on vastaketapahtuma, jossa tilakone tyypillisesti kirjoittaa tai lukee vastakkeista ja lopputuloksen perusteella päättää seuraavasta tilasta. Luku- tai kirjoitusoperaation epäonnistuessa siirrytään tyypillisesti virheenkäsittelyyn, jossa luodaan virheilmoitus asiakassovellukselle ja siirrytään tilaan "CLIENT_WRITE". Tila "WAIT_SERVER" on poikkeuksellinen, koska se voi laueta välittömästi palvelimen ollessa heti valmis tai sitten tapahtumaa voidaan myös joutua odottamaan. Odotustilanteessa tapahtuman voi laukaista palvelimen rekisteröityminen tai sitten toisen tilakoneen siirtymä tilaan "SRV_READ", jossa äskettäin varattu palvelin vapautetaan. Virhekäsittely on tässä esitetty yksinkertaistetusti, koska käytännössä jokainen operaatio voi epäonnistua, eikä ole tarkoituksenmukaista esittää yksityiskohtia jokaisesta mahdollisesta virhetapauksesta. Taulukossa 6 on kuitenkin pyritty kiinnittämään huomiota tärkeimpiin erikoistapauksiin.

Tapahtuma	Merkitys	Huomioitavaa
CS_READ_RDY	Lähettäjän vastake on valmis luettavaksi.	Tämä tapahtuma saadaan myös silloin kun TCP-yhteys on sulkeutunut.
CS_WRITE_RDY	Lähettäjän vastake on valmis kirjoitettavaksi	Mikäli TCP-yhteys on sulkeutunut, kirjoitusoperaatio voi epäonnistua
SS_READ_RDY	Palvelimen vastake on valmis luettavaksi.	Tämä tapahtuma saadaan myös silloin kun TCP-yhteys on sulkeutunut. Yhteyden sulkeutuminen tunnistetaan kun lukuyritys epäonnistuu.
SS_WRITE_RDY	Palvelimen vastake on valmis kirjoitettavaksi	Mikäli TCP-yhteys on sulkeutunut, kirjoitusoperaatio voi epäonnistua
WAIT_SERVER	Palvelin on varattu transaktiota varten	Palvelimen varaus on voinut epäonnistua. Sopivaa palvelinta ei ehkä löydy.
TIMEOUT	Tilaan sidottu tapahtuma on kestänyt kauemmin kuin asetettu aikaraja.	Asiakasta tai palvelinta ei voida odottaa loputtomasti, koska jossain vikaantumistilanteissa tämä johtaisi suhteettomaan pitkään odotusaikaan.

Taulukko 6 Tapahtumat

Taulukossa 7 esitetty ehtotaulukko kuvaa ehtoja, joita jokaisessa tilakoneen eri tapahtuman käsittelyssä määrittelevät voidaanko siirtyä seuraavaan tilaan. Ehdot määrittelevät usein myös tilan johon siirrytään, lisäksi ehto määrittelee seuraavan odotetun tapahtuman. On huomioitava, että ehdot tarkastetaan sen jälkeen kun tapahtuman määrittelemä vastaketapahtuma on suoritettu. Esimerkiksi tapahtuman "CS_READ_RDY" käsittelyssä asiakassovelluksen vastakkeesta yritetään lukea tietoja ja sen onnistuessa voidaan jatkaa normaalia suoritusta, luvun epäonnistuessa siirrytään virhekäsittelyyn.

Ehto	Selitys	Huomioitavaa
NOT_READ	Koko viestiä ei ole vielä luettu ja edellinen lukuoperaatio oli onnistunut.	
READ	Koko viesti on luettu. Tämä määritellään siten, että tilakone on jäsentänyt onnistuneesti viestin otsakesegmentissä määritellyn määrän segmenttejä.	Viestin tavupituutta ei ole määritetty protokollassa, joten tällä tekniikalla ei voida toteuttaa luontevasti protokollaa jossa seuraava pyyntö on odottamassa vastakkeen puskurissa. Tilakone voisi vahingossa lukea puskuriinsa liikaa tietoa ja saada palasen seuraavasta palvelupyynnöstä.
NOT_WRITTEN	Koko viestiä ei ole vielä kirjoitettu ja edellinen kirjoitusoperaatio oli onnistunut.	
WRITTEN	Koko viesti on kirjoitettu	Käytännössä tämä tarkoittaa sitä, että kaikki sisäisessä puskurissa olevat tavut on nyt kirjoitettu "write"-systeemikutsulla. Tämä ei tietenkään tarkoita sitä, että ne olisivat jo palvelimella tai edes lähteneet verkkoon paikallisen koneen TCP-puskurista.
ERROR	Virhetilanne, useimmiten luku- tai kirjoitusoperaatio on epäonnistunut.	Asiakassovellus tai palvelinsovellus on katkaissut TCP yhteytensä. Sama virhe saadaan jos sovellus on vikaantunut ja käyttöjärjestelmä on lopettanut sen.
RESERVED	Palvelin on saatu onnistuneesti varattua	
RESERVE_FAULT	Palvelimen varaus ei ole onnistunut	Tämä ehto saadaan käytännössä silloin, kun sopivaa palvelinta, eli pyydettyä viestiä palvelevaa palvelinta ei ole löytynyt. Jonotuksen aikarajoihin liittyvät ongelmat saadaan TIMEOUT-tapahtumasta.

Taulukko 7 Ehdot

Tilakoneen toiminnasta esimerkkinä esitetään viestin luku ja välitys palvelimelle. Toiminto käsittää tilasiirtymiä tilasta "INITIAL" tilaan "SRV_READ".

1. Tilakone on tilassa "INITIAL" ja kertoo odottavansa tapahtumaa "CD_READ_RDY". Tämä tapahtuma saadaan kun sovellus on kirjoittanut viestin välityspalvelimelle ja tieto odottaa palvelimen TCP-puskurissa.
2. Pääsilmukka saa "CD_READ_RDY" tapahtuman vastakkeen kahvalle ja kutsuu tämän tilakoneen tilafunktiota antaen sille myös tapahtuman.
3. Tilakone lukee TCP-puskurista tietoa ja yrittää samalla jäsentää saamaansa viestiä. Mikäli jäsennys jää kesken, tilakone siirtyy tilaan "CLIENT_READ" ja kertoo odottavansa tapahtumaa "CS_READ_RDY". Kaikki jo luettu tieto talletetaan tilakoneen sisäiseen puskuriin. Ennen pitkää koko viesti on luettu ja ymmärretty ja mikäli mitään ongelmia ei ilmene, siirrytään tilaan "WAIT_SRV" ja kerrotaan odotetuksi tapahtumaksi "WAIT_SERVER".

4. Palvelin vapautuu onnistuneesti ja se merkitään varatuksi tällä tilakoneelle. Tilakone siirtyy tilaan "SRV_WRITE" ja odottaa tapahtumaa "SS_WRITE_RDY".
5. Pääsilmukka saa "SS_WRITE_RDY" tapahtuman palvelimen vastakkeen kahvalle ja kutsuu tilakoneen tilafunktiota.
6. Tilakone kirjoittaa viestin TCP-puskuriin ja sen onnistuttua siirtyy tilaan "SRV_READ" ja odottaa tapahtumaa "SS_READ_RDY"

Viestin lukeminen ja välittäminen asiakassovellukselle on täysin analoginen edellisen kanssa, mutta tietysti tapahtumat kulkevat vain toiseen suuntaan.

Tilakoneisiin perustuvalla arkkitehtuurilla pyrittiin ennen kaikkea yksinkertaisuuteen, sekä mahdollisuuteen käyttää tunnettuja ja perinteisiä ratkaisuja. Toteutuksen haittapuolena on, että joistain ratkaisuista tuli verraten kömpelöitä ja keinotekoisia. Tehtävät joudutaan jakamaan mallissa pieniin palasiin, mikä ei sovi joillekin toiminnoille. Hinta on kuitenkin pieni siitä edusta, että välityspalvelimesta saatiin luotettava ja deterministinen. Jälkimmäisestä on suurta etua tutkittaessa muistivedoksia vianselvityksen yhteydessä.

5.4 Kaupalliset vaihtoehdot

Käytännössä ainoa kaupallinen vaihtoehto järjestelmälle oli BEA-Systems [BEA] yrityksen "Tuxedo"-tuote [Prim 95], joka on viestien välitykseen ja transaktioiden hallintaan tehty tapahtumamonitori. Tuxedo on käytännössä välityspalvelin lisättynä useilla erilaisilla tuotteistetuilla komponenteilla. Käytännössä tuote olisi ratkaissut ongelman varsin samalla tavalla ja jälkeenpäin ajatellen kenties paremmin, kuin itse tehty välityspalvelin. Tuxedo:n toimintalogiikka ja rajapinnat sovelluksiin ovat semanttisesti hyvin samanlaiset kuin välityspalvelimeen tuotetut. Käytännössä kuitenkin kaksi syytä painoi tarpeeksi ratkaisuun tehdä välityspalvelin itse.

Ensimmäinen syy oli TietoEnatorilla käytetty sovelluskehitysympäristö, joka on talon sisällä itse tehty C++-pohjainen oliokirjasto. Oliokirjastossa löytyy runsaasti apuoliota tyypillisen TietoEnatorin projektiohjelmiston tekoon; tietokantaluokat, merkkijonoluokat, ajankäsittelyluojat jne. Oliokirjasto muistuttaa paljon nykyistä standardia Java SDK ohjelmistoa [Java 2000], jossa ohjelmointiympäristö sisältää tavallisimmat luokat. Valmiilla luokilla voidaan nopeuttaa kehitystä, kun kaikkea perustoiminnallisuutta ei tarvitse tehdä joka kerta uudestaan. Oliokirjaston heikkous oli, että se vaati tiukasti rajatun ympäristön. Uuden tuotteen ja sen kirjastojen tuominen tähän ympäristöön nähtiin riskinä, joskaan ei suurena, koska kokemuksia "Tuxedo"-tuotteesta oli ennestään. Jälkeenpäin kävi ilmi, että riski ei ollut realisoitunut muissa projekteissa.

Toinen syy oli tuotteen hinta. Projektin kuluissa kolmannen osapuolen hankinta- ja lisenssimaksuilta välttyminen painoivat paljon ratkaisua tehdessä. Yhtenä lisäetuna nähtiin myös se, että olisi tulevaisuudessa helpompi tukea järjestelmää mitä enemmän siinä olisi toimittajan omia komponentteja.

Kehitetystä välityspalvelimesta puuttuu tärkeä "Tuxedo"-tuotteen sovellusrajapinnan ominaisuus. Ominaisuus on 2-vaihetransaktio, [Hall 96] joka mahdollistaa entistä korkeamman hajautusasteen. 2-vaihetransaktion avulla palvelimet voidaan asettaa palvelemaan entistä pienempiä viestejä ja siten yksinkertaistaa niiden rakenneta. Ominaisuus mahdollistaa järjestelmäarkkitehtuurin, jossa asiakassovellus tekee useita pyyntöjä kerrallaan ja välityspalvelin pitää huolta siitä, että jokainen transaktio tapahtuu vain jos kaikki viestit onnistuvat. Mikäli yksikin viesti epäonnistuu, välityspalvelin voi merkitä kaikki transaktiot epäonnistuneiksi. Esimerkiksi, jos asiakassovellus haluaa lisätä järjestelmään uuden käyttäjämerkinnän ja samalla lisätä hänelle roolin, voi 1-vaihetransaktiossa käydä siten, että käyttäjä saadaan lisättyä vaikka roolin lisääminen epäonnistuisi. 2-vaihetransaktiossa kummankin transaktion täytyy onnistua jotta sekä käyttäjä ja vastaava rooli lisättäisiin tietokantaan.

Välityspalvelin tukee vain 1-vaiheista transaktiota, joten asiakas- ja palvelinsovellukset luotiin siten, että ne pärjäsivät ilman esiteltyä toiminnallisuutta. Mikäli 2-vaihetransaktio oli pakollinen, tämä simuloitiin lähettämällä peruutusviesti, edellisessä esimerkissä olisi luotu käyttäjä poistettu jos roolin lisäyksen epäonnistuttua.

Tuxedo olisi tarjonnut runsaasti lisätoiminnallisuutta valvontaan ja resurssien hallintaan, mutta vaikka nämä toiminnot olisivat olleet hyödyllisiä ilman niitä tultiin lopuksi hyvin toimeen.

5.5 Prosessivalvonta

Välityspalvelin ja palvelinprosessit eivät ole virheettömiä ohjelmia, joten on todennäköistä, että ne silloin tällöin vikaantuvat. Vikatilanteen käsittelemiseksi tarvitaan automaattinen ohjelmisto, joka huomaa prosessien vikaantumisen ja tilanteen vaatiessa lopettaa ne ja käynnistää uudelleen.

Valvontatarkoitukseen tehtiin prosessienhallintaohjelmisto, joka pystyy säännöllisesti tarkastamaan prosessin statuksen pääasiassa kahdella eri tavalla. Ohjelmisto voi käyttää käyttöjärjestelmän "pr_stat" systeemikutsua ja tarkistaa onko prosessi olemassa perusteella. Vaihtoehtoisesti ohjelmisto voi lähettää palvelinprosessilla välityspalvelimen kautta tarkastusviestin, johon se odottaa vastausta annetun aikarajan sisällä. Valvontaohjelmisto toteaa palvelinprosessin terveeksi jos se saa vastausviestin. Edellisissä kappaleissa on käsitelty aikarajoihin liittyviä ongelmia palvelimen ollessa hyvin kuormitettu. Tarkastusviesti voi joutua jonoon ja jopa jäädä palvelematta, mikäli välityspalvelimen sisäinen aikaraja ylittyy. Esitetyssä tilanteessa valvontaohjelma voisi lopettaa ja käynnistää uudelleen vielä toimivan palvelinsovelluksen näin pahentaen ylikuormasta johtuvaa jonotustilannetta. Väärinkäytöksen välttämiseksi valvontaprosessille riittää, jos se saa vastausviestin ainakin yhteen kolmesta tarkastusviestistä. Näin valvontaohjelmisto ei häiritse ylikuormassa olevaa järjestelmää lopettamalla tahattomasti toimivia palvelimia.

Valvontaohjelma voisi teoriassa käyttää hyväkseen suoraan välityspalvelimelta saatuja tietoja, tällöin sen kuitenkin pitäisi tehdä säännöllisiä kyselyjä välityspalvelimelle saadakseen selville kuinka kuormattuja prosessit ovat. Toiminnallisuutta suunniteltiin mutta ei kuitenkaan koskaan toteutettu, normaali tarkastusviesti ja prosessinumeron tarkistaminen katsottiin riittäväksi ratkaisuksi ja se onkin käytännössä toiminut hyvin. Valvontaohjelma rekisteröityy myös tavallisena palvelimena välityspalvelimeen, näin sille voidaan tehdä kyselyjä prosessien tilasta viesteillä ja käynnistää eränä uusia prosesseja.

Välityspalvelimen prosessin valvomisen valvontaohjelmisto toteuttaa täysin samalla menetelmällä kuin tavallisen palvelimen. Ainoa ero on erilainen tarkastusviesti, johon välityspalvelin vastaa suoraan itse. Mikäli välityspalvelimen prosessi häviää tai välityspalvelin ei vastaa, valvontaohjelma käynnistää sen uudelleen. Palvelin- ja asiakassovelluksille ei tarvitse tässä tapauksessa tehdä mitään, koska niiden sovellusliittymät luovat automaattisesti uuden yhteyden välityspalvelimeen. Valvontaohjelma rekisteröityy tässä tilanteessa itsekin uudestaan välityspalvelimeen. Valvontaohjelma on itse immuuni reitittimen ongelmiin, joten se ei ole riippuvainen viestiyhteydestä toisin kuin tavalliset palvelimet ja asiakassovellukset.

Järjestelmän käynnistymisen yhteydessä valvontaohjelma hoitaa itse välityspalvelimen ja alkukäynnistyksen, näin se saa tietoonsa kaikkien prosessien prosessinumerot ja voi alkaa suoraan valvoa prosesseja. Järjestelmä käyttää raporttien, eräajojen ja tulostus-eräajojen käynnistämiseen prosessinvalvontaohjelmistoa. Käyttäjän käynnistäessä raportointi- tai eräajopyynnön sovellus lähettää viestin valvontaohjelmiston palvelimelle, joka joko käynnistää uuden eräajon tai sitten raportoi virheellä käynnistymisen epäonnistuttua. Järjestelmän kaikki ajettavat prosessit on kuvattu rakenteelliseen konfigurointitiedostoon, johon voidaan merkitä myös rajoitukset ja muut prosesseihin liittyvät parametrit, kuten onko uudelleen käynnistäminen vikaantumistilanteessa sallittua.

Valvontaohjelmiston toimintaan ei paneuduta tässä syvällisemmin, rakenteeltaan se muistuttaa paljon välityspalvelimen arkkitehtuuria. Valvontaohjelmiston tekee kuitenkin paljon monimutkaiseksi

pääsilmukan toteutus, joka joutuu käyttämään välityspalvelimen synkronista sovellusrajapintaa ja jonka pitää silti pystyä ajamaan tilakoneita ajastetusti.

5.5.1 Raportointi

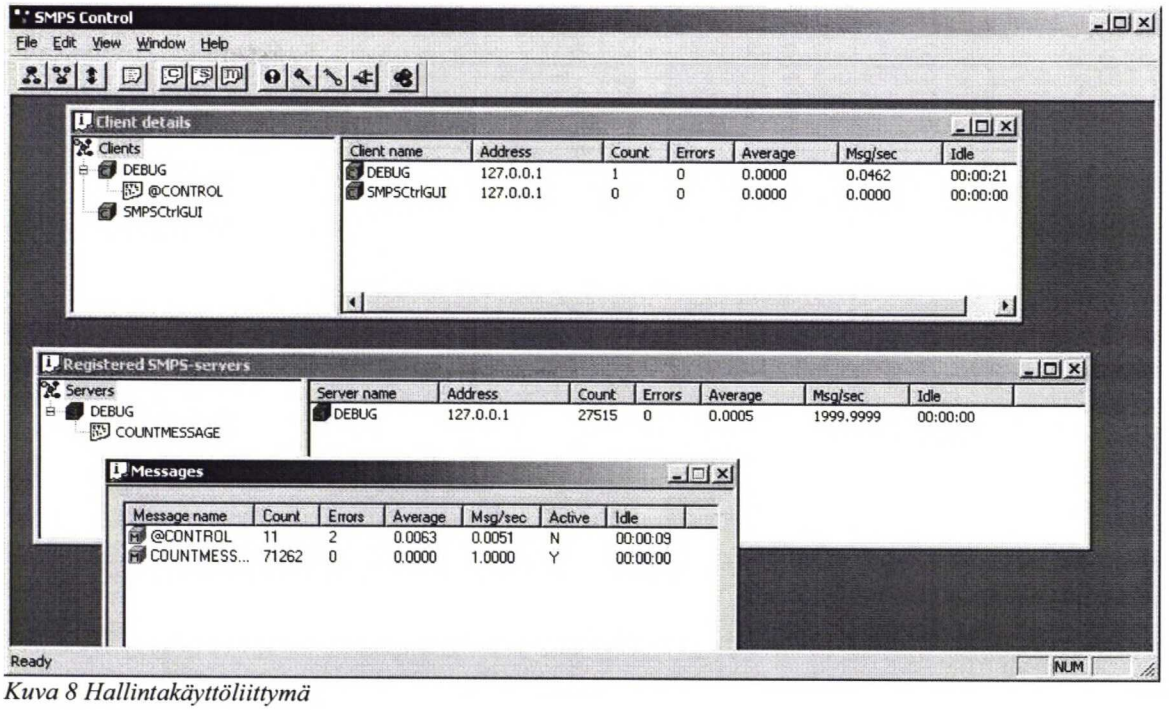
Valvontaohjelmisto kykenee raportoimaan prosessien ongelmista asetettuun sähköposti-osoitteeseen. Ratkaisu olettaa, että sähköpostipalvelut ovat toiminnassa, joten esimerkiksi järjestelmän verkko-ongelmissa toiminto on valitettavasti hyödytön. Sähköposti lähetään muutamaa yksinkertaista sääntöä käyttäen. Prosessi vikaantuessa ensimmäisen kerran, lähetetään sähköposti raportoiden kaikki prosessit, jotka ovat vikaantuneet edellisellä tarkastusvälillä. Mikäli prosessia ei saada uudelleen käyntiin monista yrityksistä huolimatta, lähetetään vielä yksi sähköposti jossa todetaan ettei prosessin käynnistys ole onnistunut ja sen uudelleenkäynnistysyrityksistä luovutaan.

Järjestelmänvalvojan työtä helpotetaan minimoimalla sähköpostien määrä, joten valvojan on paljon helpompi huomata ongelmatilanteet. Automaattisella sähköpostiviestillä voidaan nopeasti reagoida ongelmiin, jotka ovat suhteellisen harvinaisia mutta estävät koko järjestelmän toiminnan. Järjestelmän rampauttavia ongelmia ovat tyypillisesti levyn täyttyminen ja sen välilliset seuraamukset. Järjestelmänvalvoja usein käytännössä jättää ensimmäisen sähköpostin huomiotta ja reagoi vasta jos seuraava, uudelleenkäynnistymisen epäonnistumisesta, ilmoittava sähköposti saapuu. Käytäntö mahdollistaa nopean reagoinnin koko järjestelmää uhkaaviin ongelmiin eikä järjestelmänvalvojan tarvitse huolehtia järjestelmän aktiivisesta tarkkailusta.

5.6 Hallintakäyttöliittymä

Järjestelmää varten tarvittiin hallintakäyttöliittymä, jolla voitaisiin saada välityspalvelimelta reaaliaikaiset tiedot viestien tiloista ja palvelimien tilastoista. Hallintaliittymällä olisi myös tarkoitus nähdä mitkä palvelimet olivat rekisteröityneet välityspalvelimeen ja mitä viestejä ne palvelivat. Tärkeä tieto viesteistä oli myös palvelimien viestinpalveluviive, josta piti myös saada päivitetty tieto. Hallintakäyttöliittymästä piti myös nähdä eri asiakassovellukset ja mistä ne välityspalvelimeen olisivat yhteydessä. Asiakassovelluksen tietoihin tarvittiin myös IP-osoite ja käyttäjätunnus, josta nähtäisiin mistä työasemasta asiakassovellus oli yhteydessä ja kuka käyttäjä sovellusta käyttää.

Hallintakäyttöliittymä päätettiin toteuttaa Microsoft Windows alustalle käyttäen MFC [Jones 99] rajapintaa, jolla voitiin luoda käyttöliittymä ja esittää järjestelmä puumaisessa muodossa. MFC rajapinta tarjoaa nämä käyttöliittymäelementit valmiina standardikomponentteina.



Kuva 8 Hallintakäyttöliittymä

Hallintakäyttöliittymä käyttää välityspalvelimeen samaa viestiyhteyttä kuin muutkin sovellukset, mutta useimmat sen käyttämät viestit ovat välityspalvelimen itse palvelemia, eikä niitä reititetä mihinkään. Sisäisiä viestejä käsitellään välityspalvelimessa hallintaviestien tilakoneilla.

Hallintakäyttöliittymä lähettää myös viestejä prosessivalvontaohjelmistolle, joilla se saa tiedot palvelinprosessien tiloista. Samoja viestejä käyttäen hallintaliittymällä voidaan myös käynnistää ja lopettaa prosesseja, mutta käytännössä tämä ei ole osoittanut tarpeelliseksi kuin joissain harvoissa poikkeustilanteissa.

Hallintaliittymästä voidaan saada esille seuraavat tiedot kukin omassa taulukkoikkunassaan.

- Palvelimien nimet, palveltavat viestit, Palvelimen IP-osoitteet ja keskimääräinen viestin vastausviive
- Asiakassovelluksen nimet, IP-osoitteet, käyttäjätunnus ja keskimääräinen viestin vastausviive.
- Viestit, viestien määrät ja niiden keskimääräinen vastausviive.
- Ajossa olevat prosessit mukaan lukien eräajot käynnistysparametreineen.

Tarjotuista tiedoista voidaan melko helposti arvioida järjestelmän tila, varsinkin kun jokaisen ikkunan tiedot voi lajitella vapaavalintaisen sarakkeen mukaan. Lisäksi hallintaliittymä tukee välityspalvelimen asettamista huoltotilaan. Huoltotilassa välityspalvelin vastaa kaikkiin sovellusten lähettämiin viesteihin järjestelmänhallitsijan asettamalla viestillä, jonka sovellusohjelmat näyttävät dialogi-ikkunana. Ominaisuus tehtiin, koska useimmiten suurin osa käyttäjistä ei joko saanut tietoa päivityksestä tai ei sitten muistanut päivitysaikaa. Näin voidaan estää käyttäjien hämääntyminen palvelukatkoksessa kun palvelinkomponentteja päivitetään. Huoltotoimenpiteet suoritettuaan järjestelmänhallitsija voi kytkeä välityspalvelimen normaaliin toimintaan, jolloin tuotanto jatkuu kuten ennen katkosta. Hallintaliittymästä voidaan myös kytkeä viestien jäljitys päälle ja ja saada erityisen jäljityspalvelimen avulla levylle kirjoitetut viestit näkyville. Tämä on kuitenkin täysin turha ominaisuus,

koska on paljon helpompaa vain tutkia tiedostoja levyltä jonne välityspalvelin ne on kirjoittanut.

Graafisen hallintakäyttöliittymän merkitys on käytännön tuotannossa merkityksetön eikä sitä käytetä lainkaan. Sitä vastoin samat toiminnot tarjoava komentoriviohjelma on huomattavasti käyttökelpoisempi, koska se voidaan helposti integroida skripteihin joilla järjestelmää ohjataan. Komentoriviohjelma käyttää täysin samoja viestejä kuin graafinen hallintakäyttöliittymä.

Skriptien ympärille on rakennettu valikko, jolla voidaan helposti valita tavallisia hallintaoperaatioita kuten palvelimien käynnistämistä, lopettamista ja tilakyselyjä. Tilakyselyt rajoittuvat lähinnä tietoon siitä, ovatko palvelimet päällä vai ei. Valikosta voidaan myös käynnistää helposti viestien jäljitys. Tämä on erittäin käytännöllistä koska jäljitystiedostot kirjoitetaan saman palvelimen levylle, joten niitä on helppo seurata reaaliaikaisesti tiedostoista käyttäen esimerkiksi "less"-komentoa. Ohessa on esimerkki jäljitystiedostoista. Lähetetyille ja vastaanotetuille viesteille on omat tiedostonsa, jotka on merkitty ".in" ja ".out" päätteillä. Tiedoston nimi määräytyy käytetystä suodatuskriteeristä.

mtdebugsrv.in

```
21:36:07 >
CTL;16;;21:36:07;DEBUG;;;1;COUNTMESSAGE;N;N;0;Y;;;;FREE;3;Req:
350;

21:36:07 >
CTL;16;;21:36:07;DEBUG;;;1;COUNTMESSAGE;N;N;0;Y;;;;FREE;3;Req:
351;
```

mtdebugsrv.out

```
21:36:07 >
CTL;16;;21:36:07;mtdebugsrv;;;0;COUNTMESSAGE;N;N;0;Y;;;;

21:36:07 >
CTL;16;;21:36:07;mtdebugsrv;;;0;COUNTMESSAGE;N;N;0;Y;;;;
```

5.7 Kannettavuus

Tilakonemalliin perustuva arkkitehtuuri on helppo siirtää käyttöjärjestelmästä toiseen sillä käytetyt periaatteet löytyvät kaikista käyttöjärjestelmistä. Välityspalvelin on tehty käyttäen POSIX määrittelyn mukaisia systeemikutsuja, nämä ovat saatavilla sekä UNIX- että Windows alustalle. Välityspalvelin pystyttiin siten helposti kääntämään myös Microsoft Windows-alustalle, joskin tämän tarkoitus oli vain kokeellinen.

Alun perin HP-UX alustalle tehty koodi oli vaivatonta siirtää POSIX-yhteensopivuuden ansiosta Linux-järjestelmään [Linux] joskin tiettyjä ongelmia havaittiin. Käytetty Linux-ydin 1.2.17 sisälsi virheen, jossa vastakkeiden hyväksymiseen käytetty "accept"-systeemikutsu toimi väärin palvelimen kuormittuessa. Ytimen vikaannuttua systeemikutsu ei enää jossain vaiheessa palauttanut yhteyttä muodostavan asiakaskoneen IP-osoitetta, vaan antoi sattumanvaraisia tavuja tietorakenteeseen.

Muutoin kannettavuudessa tuli pieniä ongelmia vain Microsoft Windows-alustan ja HP-UX alustan eroista, jotka eivät kuitenkaan olleet merkittäviä. Käytännössä pienet erot Berkeley Socket-rajapinnassa ja ajankäsittelyfunktioiden käytössä vaativat alustakohtaisen toteutuksen. Sovelluskirjastojen siirtoja kääntäminen eri alustojen välillä oli helppoa.

Prosessivalvonta-ohjelmaa ei ollut mahdollista siirtää Windows-maailmaan erilaisen prosessihallinnan rajapintojen takia, mutta siihen ei toisaalta ollut mitään tarvetta.

Käytännössä hyvä kannettavuus oli välityspalvelimelle turha ominaisuus, mutta sovellusliittymän kirjastoille se oli välttämätön.

5.8 Yhteenveto

Välityspalvelinta suunniteltaessa oli alusta lähtien selvää, että luottavuus ja yksinkertaisuus ovat pääasiallisia suunnittelukriteereitä. Luotettavuus voidaan saavuttaa useammalla tavalla, joista toiminnallisuuden yksinkertaisuus on yleensä paras menetelmä. Mitä yksinkertaisempi ohjelmisto on, sitä helpompi se on testata ja sitä vähemmän sillä on tiloja. Yksisäikeinen tilakonemalliin perustuva rakenne vastaan monisäikeisen malli on vain näennäisesti monimutkaisuuden suosimista yksinkertaisuuden kustannuksella. Käytännössä monisäikeinen ohjelma on huomattavasti hankalampaa tehdä siten, että se toimii oikein kaikissa tilanteissa, johtuen suuremmasta määrästä ohjelman mahdollisia tiloja. Yksisäikeisessä ohjelmassa itse toteutus ja suunnittelu on hankalampaa, koska kaikki toiminnot pitää määritellä tilakoneilla ja näiden tilat määritellä sitten, että ne ovat mahdollisimman nopeita eikä järjestelmän reaaliaikaisuus häiriinny.

UNIX-järjestelmät tarjoavat paljon esimerkkejä vastaavista ratkaisuista, kuten edellä on mainittu. Välityspalvelimen arkkitehtuuri on jossain määrin kopio tyypillisestä WWW-palvelimesta jonka täytyy palvella satoja yhtäaikaisia käyttäjiä yhdellä prosessilla. Tämän vuoksi pidettiin varmana, että toteutuksen yhteydessä ei tulisi mitään ikäviä yllätyksiä, ainakaan perusarkkitehtuurin osalta.

6 Toteutus

Kappale esittelee välityspalvelimen toteutuksen vaiheita siinä määrin kuin niillä on merkitystä tämän työn kannalta. Suunnitteluun ja toiminnallisuuden esittelyyn palataan vain silloin, kun toteutuksessa huomattiin jokin suunnitteluvalinta virheeksi tai sitä jouduttiin muuttamaan. Kappaleessa esitellään myös eräitä toteutuksen kiinnostavimpia ongelmia.

Ohjelmiston toteutuksessa moni asia oli jo lyöty lukkoon, joten mitään monimutkaisia ratkaisuja ei tultaisi tarvitsemaan. Ohjelmointikieleksi valittiin C++, koska siinä tietorakenteiden teko on helppoa ja tilakoneet voidaan esittää olioina, joissa on jäsenmetodi jokaista tilafunktiota kohden. Tilakoneoliot voivat myös ylläpitää luontevasti tilaansa, luku- ja kirjoituspuskureita sekä muita apumuuttujia luokan jäsenmuuttujina.

Asiakas- ja palvelinsovellusten ohjelmointikieli oli myös C++, joten välityspalvelimen sovelluslittymäkirjastot piti tuottaa olioilla. Uudelleenkäytettävyyden maksimoimiseksi olio-pohjainen rakenne oli myös edullinen, kun sekä välityspalvelin että sovelluslittymät pystyivät uudelleenkäyttämään samoja luokkia. Välityspalvelimen toteutuksena puhtaasti ANSI C-kielinen ohjelmakin olisi ollut mahdollinen ja jälkeenpäin ajateltuna kenties jopa parempi vaihtoehto. Tämä lähinnä sen vuoksi, että välityspalvelimen rakenne ei tarvitse oliomallinnusta. Nykyisellään toteutus olikin perinteisen proseduraalisen C-ohjelman [Appel 97] ja olio-pohjaisen C++-ohjelman sekoitus. Pääsäie oli luontevinta tehdä tavallisen C-ohjelman tapaan jossa itse tilakoneet on toteutettu olioilla.

Aputietorakenteina tarvittiin luokat linkitettyä listaa ja hajautustaulukkoa varten [Corm 96]. Kuten suunnittelua käsiteltäessä todettiin, järjestelmä ei tarvitsisi hajautustaulukkoa tietorakenteenaan. Hajautustaulukko kuitenkin toteutettiin, mutta vain mielenkiinnon vuoksi. Periaatteessa alustan standardeja C++ template-kirjastoa [ISO14882] olisi voitu käyttää yhtä hyvin. Toteutuksen yhteydessä yllätti, kuinka vaikeaa edellä mainittujen perustietorakenteiden teko oli niiden yksinkertaisuudesta huolimatta.

6.1 Välityspalvelin

Välityspalvelin on järjestelmän ydinkomponentti, joka huolehtii viestien välityksestä. Tilakoneiden valinta ohjelman toteuttamiseksi osoittautui tähän tarkoitukseen erittäin hyväksi ratkaisuksi, koska tilakonetaulukko oli suoraviivaista kirjoittaa sellaiseen koodiksi. Kukin tilakoneen siirtymä oli helppo toteuttaa pala kerrallaan rakentaen jokaiselle tilalle oma tilafunktio. Tilakonemalliin perustuvassa rakenteessa ei tarvitse huolehtia tietorakenteiden synkronoinnista, joten tilakoneiden pääsy erilaisiin jaettuihin tietorakenteisiin oli helppo tehdä sivuvaikutuksettomasti.

Käynnistyessään palvelin lukee konfiguraationsa tiedostosta, joka määrittelee muutamia peruskonfigurointiparametreja, kuten lokihakemiston ja kuunnellun TCP-porttinumeron. Konfiguraation määrittelemät arvot voidaan antaa myös komentoriviltä, joka mahdollistaa ohjelman luontevan integraation hallintaskriptiin. Komentorivin parametrien jäsennys toteutettiin "getopt"-kirjastolla. Käynnistyttyään ja kelpuutettuaan konfiguraation, välityspalvelin yrittää avata paikallisen TCP-portin ja sen onnistuttua siirtyy pääsilmukkaan. Pääsilmukan alussa luodaan vakiotilakoneet ja ajastetaan ne. Käytännössä välityspalvelimen konfiguraatiotiedostoa ei koskaan käytetty, koska oletusarvoiset parametrit olivat riittäviä tai sitten arvot voitiin antaa komentoriviltä.

Välityspalvelimen pääsilmukka on täysin riippuvainen "select"-systeemikutsusta, jolle välityspalvelin on antanut tarkasteltavaksi kaikki vastakekahvat joista se tarvitsee tapahtumia. Palvelinvastake on aina tässä yhteydessä annettu systeemikutsun argumentiksi. "select"-systeemikutsu palauttaa numeron, joka kertoo kuinka monelle vastakkeelle on odotettu tapahtuma valmiina. Paluuarvo ollessa nolla tarkoittaa se, että systeemikutsulle annettu aikaraja on mennyt umpeen ennen kuin yhtään vastaketapahtumaa saatiin. Tästä alkaa pääsilmukan suorituskerta, joka palaa lopuksi "select"-systeemi-

kutsuun odottamaan uusia tapahtumia.

Pääsilmukka tarkistaa aina ensiksi onko palvelinvastake lukuvalmis. Tämä tarkoittaa sitä, että välityspalvelimen kuuntelemaan TCP-porttiin on tullut uusi yhteys jonka käyttöjärjestelmä on hyväksynyt. Mikäli palvelinvastake on lukuvalmis, pääsäie kutsuu "accept"-systeemikutsua ja hyväksyy uuden yhteyden. Systeemikutsu palauttaa hyväksytyn vastakkeen kahvan, jonka pääsäie voi antaa uudelle tilakoneelle. Ennen uuden tilakoneen luomista pääsäie tarkistaa onko yhtäaikaisten yhteyksien ja transaktioiden määrä sallitun rajan sisällä, jos näin on, luodaan uusi tilakone ja alustetaan se "INITIAL" tilaan. Mikäli transaktioiden määrä on jo säädetyllä ylärajalla, yhteys suljetaan välittömästi.

Toteutuksen yhteydessä huomattiin tilakoneen aiheuttama kisatila, (eng. "race condition") joka saattoi aiheuttaa pysyvän ongelman seuraavassa skenaariossa.

1. Järjestelmä on raskaassa käytössä ja jotain paljon käytettyä viestiä palvelee muutama palvelin kuormaa jakaen. Raskaasti kuormatussa palvelimessa viestin palvelu myös kestää useita sekunteja.
2. Yksi palvelimista vikaantuu, ja koska se ei enää palvele viestejä, jäljelle jääneille palvelimelle alkaa kerääntymään jonoa.
3. Prosessinvalvontaohjelmisto käynnistää vikaantuneen palvelimen uudelleen
4. Palvelin yrittää rekisteröityä, mutta ei onnistu, koska jonottamaan jääneiden viestien vuoksi sisäinen transaktiotaulukko on täynnä.
5. Palvelin ei onnistu rekisteröitymään ja lopettaa suoritukseen. Sykli jatkuu sitten kohdasta 3, prosessinvalvontaohjelmiston käynnistäessä palvelimen uudestaan.

Kisatila ajaa välityspalvelimen tilanteeseen jossa kuorma ei koskaan hellitä, koska uutta palvelinta ei saada rekisteröitymään välityspalvelimeen jakamaan kuormaa. Käytännössä ongelma voitiin kiertää kasvattamalla transaktioiden enimmäismäärää merkittävästi, sillä vaatimusten määrittelemä sata yhtäaikaista transaktiota oli hyvin matala luku. Täytyy muistaa, että kun tiedetään asiakassovellusten ja palvelinsovellusten määrä, voidaan aina sanoa avoimien transaktioiden enimmäismäärä. Tämä enimmäismäärä on asiakassovellusten määrä summattuna palvelinsovellusten määrään. Palvelinsovelluksen lasketaan mukaan, koska myös ne voivat avata uuden transaktion lähettämällä uuden viestin. Jokainen asiakassovellus voi pitää yllä vain yhtä transaktiota kerrallaan (monisäikeisiä asiakassovelluksia ei käytetty) ja palvelinsovellus voi sekä palvella että ylläpitää yhtä transaktiota kerrallaan. Yhtäaikaisten transaktioiden enimmäismäärä on helppo määritellä muutamaan sataan.

Taulukossa 8 on esitetty välityspalvelimen ohjelmistokomponentit.

Komponentti	Selitys	Huomautuksia
smpsv2kernel	Ajettava ohjelma. Itse välityspalvelin.	Linkitetty käyttämään sovelluslittymäkirjastoa
smpsv2control	Ajettava ohjelma Komentorivipohjainen hallintaohjelma.	Linkitetty käyttämään sovelluslittymäkirjastoa

Taulukko 8 Välityspalvelimen komponentit

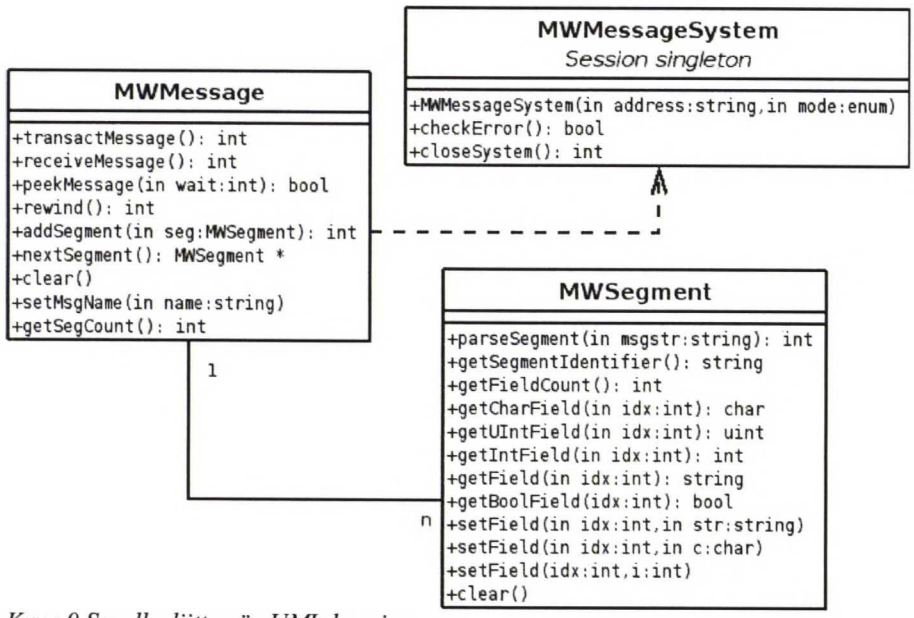
Toteutuksen yhteydessä huomattiin tarve yleislähetysviesteille (eng. "broadcast message"), jolla voitaisiin kaiuttaa kaikille palvelimille lähes yhtä aikaa sama tieto. Moneen palvelimeen ohjattu viesti ei sopinut tilakoneen transaktiomalliin, koska nythän yhtä pyyntöviestiä vastaisi lukuisia vastausviestejä. Välityspalvelimen tilakonetoteutukseen jouduttiin tekemään muutos. Yleislähetysviesti on asiakassovelluksen ja palvelinsovelluksen näkökulmasta tavallinen viesti. Tilakone tunnistaa

yleislähetysviestin nimestä "@BROADCAST" ja lähettää sen jokaiselle palvelimelle yksi kerrallaan siirtyen seuraavaan aina saatuaan vastauksen. Asiakassovelluksen avaamaan transaktioon voi tulla vain yksi vastaus, joten tilakone ei huomioi palvelinten vastauksia. Tilakone kuittaa yleislähetyksen vakioviestillä asiakassovellukselle, kun kaikki palvelimet on käyty läpi. Mikäli lähetys jollekin palvelimelle epäonnistuu, asiakassovellus ei saa tästä mitään tietoja.

Yleislähetysviestiä käyttävät erityiset konfiguraatiopalvelimet, joiden tietoja monet muut palvelimet pitävät omassa välimuistissaan. Perustiedon muuttuessa, konfiguraatiopalvelin pystyi yleislähetysviestillä kertomaan sen kaikille muille palvelimelle. Yleislähetysviestiä ei voinut suunnata vain tietylle joukolle palvelimia. Palvelimet, jotka eivät olleet kiinnostuneita yleislähetysviesteistä, toteutettiin siten, että jättävät yleislähetysviestin huomiotta ja vastaavat tyhjällä paluuviestillä..

6.2 Liittymäkirjastot

Asiakas- ja palvelinsovellusten liittymäkirjastot toteutettiin luokkakirjastoksi, jossa on kolme pääluokkaa. Näitä pääluokkia ovat istuntoa, viestiä ja segmenttiä kuvaavat luokat. Sovelluksella on vain yksi instanssi istuntoa kuvaavasta luokasta ja yksi instanssi sekä pyyntö- että vastausviestiä kohden. Segmenttien instansseja voi olla useita, eivätkä ne ole riippuvaisia muista luokista. Kuvassa 9 on esitetty liittymäkirjaston UML-kaavio.



Kuva 9 Sovellusliittymän UML-kaavio

Liittymäkirjastot tehtiin sekä HP-UX, että Microsoft Windows alustalle jaettuina kirjastoina. Jaetut kirjastot mahdollistava sovellusliittymän päivityksen ilman että sovellusta tarvitsee kääntää uudestaan, ne myös säästävät resursseja kun käyttöjärjestelmän tarvitsee ladata kirjasto vain kerran muistiin. Projektin yhteydessä siirryttiin käyttämään myös järjestelmän muissa sovelluksissa jaettuja kirjastoja, ja monien sovellusten koko pieneni samalla 4 megatavusta 200 kilotavuun. Liittymäkirjasto toteutettiin myös Microsoft Windows ActiveX COM-komponenttina [Chap 96] sekä kokeiluna Java JDK 1.2.0-alustalle. Java-alustalle toteutettiin myös salaus "Blowfish"-algoritmilla [Sch 95] TCP-yhteyden tasolla. Salausta ajateltiin käytettävän selainten Java-Applet sovelluksissa tietojärjestelmään mutta tästä ajatuksesta luovuttiin.

Liittymäkirjaston rakentamisessa helpotti merkittävästi se, että toteutukseen ei tarvittu mitään sovelluslogiikka. Rakennetta yksinkertaisesti myös viestien rakenne, kaikki välitettävät viestit olivat oleellisesti puhdasta tekstiä, eikä mitään erikoisia purkukirjastoja tarvittu. Kaikki alustat tarjoavat myös valmiiksi toteutuksen "deflate"-pakkaukseen, kuten tässä liittymäkirjastossa käytetyn zlib-kirjaston [ZLIB].

Komponentti	Selitys	Huomautuksia
lib3tier	Yleiskirjasto tietorakenteille ja muille apuluokille	Käytetään myös välityspalvelimessa
libsmpsv2	Viestikirjasto	Toiminta osin riippuvainen konfiguroitavista ympäristömuuttujista.

Taulukko 9 Sovellusliittymäkomponentit

Listauksessa 6 on esitetty asiakassovelluksen esimerkkikoodia.

```
MWMessage *m = new MWMessage();

m->addSegment(reqseg1);
m->addSegment(reqseg2);

m->transactMessage(); // lähetä viesti

while(seg = m->nextSegment()) {
  // käsittele segmentti
}
```

Listaus 6 Palvelinsovelluksen esimerkkilistaus

Listauksessa 7 on esitetty palvelinsovelluksen esimerkkikoodia. Esimerkissä näkyy kuinka palvelin voi itsekin lähettää viestin.

```
MWMessage *m = new MWMessage();
MWMessage *m2 = new MWMessage();

while( 1 ) {
    m->receiveMessage(); // vastaanota pyyntöviesti

    m2->setMsgName("SOMEMSG");
    m2->addSegment(seg);
    m2->transactMessage(); // lähetä pyyntö
```

```
    // iteroi segmentit ja lisää pyyntöviestiin segmenttejä
    m->transactMessage(); // lähetä vastaus
}
```

Listaus 7 Palvelinsovelluksen esimerkkilistaus

Sovellusliittymän luokat vaativat, että sovellusohjelma iteroi jokaisen pyyntöviestin segmentin yksi kerrallaan. Tämä käytäntö ei osoittautunut parhaaksi mahdolliseksi, koska iteraattori ei ollut kovinkaan tuttu käsite sovellusohjelmoijille. Parempi ratkaisu olisi ollut tarjota viestin segmentit taulukossa. Sovellusliittymään ei kuitenkaan tehty mitään muutoksia, koska haitta katsottiin pienemmäksi, kuin vaiva muuttaa jo tehdyt ohjelmat.

6.3 Prosessinvalvonta

Prosessinvalvontaohjelmistoa varten oli käytössä runsaasti käyttöjärjestelmäkutsuja. Näillä kutsuilla on erittäin helppoa saada prosessin tila prosessinumeroa käyttäen. Käyttöjärjestelmä tarjosi myös valmiit rajapinnat prosessien lopetukseen ja käynnistykseen Toteutuksen yhteydessä havaittiin myös ongelmia, joita käsitellään tarkemmin omassa kappaleessaan.

Valvontaohjelmiston toteutuksen pääpaino keskittyi sen palvelinprosessiosioon, eli hallintaliittymälle tarjottuihin tietoihin. Kuten edellä on mainittu, valvontaohjelmisto rekisteröityi välityspalvelimeen ja tarjosi viestejä käyttäen tilastoja ja tilatietoja palvelimista sekä eräajoista. Tämän lisäksi välitysohjelmiston piti myös tarkistaa säännöllisin väliajoin prosessien tila käyttäen edellä mainittuja systeemikutsuja. Listauksessa 7 käy ilmi, että sovellusliittymä odottavan palvelinsovelluksen täytyy odottaa vastaanottavassa metodissa, kunnes välityspalvelin lähettää sille viestin. Valvontaohjelmisto haluttiin kuitenkin toteuttaa yksisäikeiseksi, kuten välityspalvelin itse. Tämä onnistui muuttamalla sovellusrajapintaa hieman siten, että siltä voi saada TCP-yhteyden vastakkeen kahvan. Näin samaa "select"-systeemikutsua voitiin käyttää ja kutsua sovellusliittymän vastaanottometodia vain silloin kun nähtiin että viesti oli tulossa. Systeemikutsun ajastuksella voitiin myös toteuttaa säännöllisin välein tehtävä tarkastus.

Järjestelmävalvojalle tehtävä sähköpostin lähetys toteutettiin suoralla SMTP toteutuksella [RFC821], joka ottaa yhteyden konfiguroituun postipalvelimeen ja lähettää viestin. Toteutuksella vältyttiin ulkoisilta riippuvuuksilta, sama toiminnallisuus olisi saatu aikaan esimerkiksi palvelimen "mail" komentoriviohjelmalla tai jollain valmiilla kirjastolla.

Taulukossa 10 on esitetty valvontaohjelmiston komponentit.

Komponentti	Selitys	Huomautuksia
projectrld	Valvontaohjelmisto	Käyttää hallintaviestejä välityspalvelimen käskyttämiseen.
projectrlc	Valvontaohjelmiston komentorivirajapinta	Käskyttää valvontaohjelmistoa välityspalvelimen kautta lähetetyillä viesteillä.

Taulukko 10 Prosessinvalvonnan komponentit

Valvontaohjelmistolla on monimutkainen konfiguraatiotiedosto, jonka lukemiseksi tehtiin jäsennin kehitysympäristön "lex" ja "yacc" jäsenningenerointityökaluilla [Appel 97]. Ohessa ote konfiguraatiotiedostosta, se on pieni osa koko konfiguraatiotiedostoa ja sisältää yleisiä SMTP-asetuksia valvontaviestiä varten, sekä ryhtyy määrittelemään erästä "KOVA"-alijärjestelmää.

```
smtp_host localhost    # mail settings
smtp_port 25
smps_address localhost:12345

system_def KOVA {
      # system settings
      user_id cckova                              # user to start the process
      system_path /mnt/contstore/kova/scripts # default home path
      notify Teemu.Ikonen@tieto.com                 # notify mail address

      log_path /mnt/contstore/kova/log
      env_script "/mnt/contstore/kova/setup/environment kova"
      description "KOVA - Container storage system"

      # define server process
      server_def KOVASERVER {
      executable_path /mnt/contstore/kova/bin/koreffserver
      initial_instances 2
      description "Serves port-program and reference GUI"
            listens STEVEREFF,              # listens these messages
                    TRCOMPANY,
                    TRUNIT,
                    CONTVISIT,
                    SECURITYCARD
            depends_of PARAMETER,           # uses these messages
                       IEERRORCODE,
                       IELOG
     }
     abstract_def KOSPCSEND {       # define server template
       executable_path /mnt/contstore/kova/bin/kospcsend
       max_number_of_instances 1
     }
     daemon_def KOSPCSEND-CHANGEDVISIT extends KOSPCSEND {
      command_line CHANGEDVISIT
     }
```

```
daemon_def KOCMISEND {
  executable_path /mnt/contstore/kova/bin/kocmisend
}
...
```

Konfiguraatiotiedosto määrittelee järjestelmän ympäristön, siinä käytetyt palvelimet sekä niiden väliset viestiriippuvuudet. Näin valvontaohjelmisto käynnistää kaikki prosessit varmasti oikein.

6.4 Hallintaliittymä

Hallintaliittymä perustuu muutamaan välityspalvelimen sisäänrakennettuun viestiin, joita itse välityspalvelin palvelee. Rakenteeltaan viestit eroavat tavallisista palveluviesteistä vain nimeltään, joten välityspalvelimeen toteutettiin yksinkertainen virtuaalipalvelin, jonka hallintakäyttöliittymän tilakoneet toteuttivat.

Hallintaliittymän eri viestityypit ovat tilasto- ja varsinaiset hallintaviestit. Tilastoviestit antavat tietoa asiakassovelluksista ja palvelinsovelluksista, joko kaikista tai yhdestä. Haku tapahtuu nimen perusteella, tavallisista viesteistä voidaan tehdä myös samanlainen kysely. Hallintaviestit ovat käytännössä palvelimien rekisteröitymistä ja poistamista varten. Poistamista tarvitaan, kun halutaan varmistaa, että palvelin on varmasti poistettu reitittimen rekisteristä. Välityspalvelin ei huomaa palvelimen poistumista ennen kuin sille yritetään lähettää viestiä. Kappaleessa 6.1 mainitussa tapauksessa tarvittiin hallintaviesti palvelimen pakotettua poistamista varten.

Hallintaa varten tehtiin kaksi ohjelmaa, jotka kummatkin toimivat käytännössä täysin identtisellä tavalla, mutta eri ympäristöissä. Komentorivipohjainen ohjelma, jotka voidaan käyttää sekä Windows että UNIX ympäristössä, sekä graafinen Microsoft Windows käyttöliittymä, jota voidaan käyttää vain Windows-ympäristössä. Myöhemmin huomattiin, että oli käytännössä turhaa tehdä graafinen käyttöliittymä, mutta tämä oli allekirjoittaneelle arvokas oppimiskokemus. Graafinen liittymä myös helpotti valvontatarpeiden ymmärtämistä, koska sitä oli helppoa esitellä eri ihmisille.

Oheisessa esimerkissä on esitetty komentorivipohjaisen ohjelman optiot.

```
roamer $ ./smpscontrol
usage: smpscontrol [options]

Options:
        -h          This screen
        -o          Set SMPS to service mode
        -r [msg]    Set SMPS to service mode with message for users
        -f          Set SMPS to normal mode
        -x          Exit smps cleanly
        -k          Kill smps without cleanup
        -d [name]   Drop server 'name'
        -p          Ping server
        -l [level]  Set the loglevel, valid values are:
                        DEBUG   - Debug information
```

```
                    INFO    - Nice to know information
                    WARNING - Warnings
                    ERROR   - Errors
                    FATAL   - only Fatal errors
                    NONE    - No logwriting
-t [name]  Start trace of client/server
-e [name]  Stop trace of client/server
-q         SMPS Process queues
-y addr    Allow clients from address
-z addr    Deny clients from address
-i -m [name] Statistics about message
-i -c [name] Statistics about client
-i -s [name] Statistics about server
-i -c all  Statistics about all clients
-i -s all  Statistics about all servers
-i -m all  Statistics about all messages
-i -u      Statistics about all address access
-a [address] Address to connect
```

Taulukossa 11 on esitetty hallintaliittymän komponentit.

Komponentti	Selitys	Huomautuksia
smpsctrlguiv2.exe	Graafinen käyttöliittymä Microsoft Windows alustalla.	Keskustelee sekä valvontaohjelmiston että itse välityspalvelimen kanssa viesteillä.
smpscontrolv2	Komentorivipohjainen käyttöliittymä kaikille alustoille.	Keskustelee vain välityspalvelimen kanssa.

Taulukko 11 Hallintaliittymän komponentit

Microsoft Windows pohjainen hallintaliittymä oli huomattavan vaikeaa toteuttaa, erityisesti ikkunoiden käsittelyn ja sulavan käyttöliittymän toteuttaminen oli erityisen vaikeaa. Arvostan kuitenkin osaamista jonka kokemuksesta sain ja käytinkin graafiseen käyttöliittymän tekoon runsaasti omaa aikaa.

6.5 Toteutuksen ongelmia

Normaalisti ohjelmiston kehityksessä tehdään runsaasti virheitä jotka korjataan lähes välittömästi. Silloin tällöin toteutuksen yhteydessä havaittiin ongelmia, joihin ei oltu varauduttu suunnittelussa tai jotka aiheuttivat ylimääräistä työtä. Kaikki ongelmat, kunhan ne oli ensin ymmärretty, olivat suhteellisen helppoja ratkaista. Ainoastaan silloin, kun ratkaisu ei sopinut arkkitehtuuriin, jouduttiin näkemään enemmän vaivaa.

Tyypillisin vaikeaselkoinen ongelma oli muistivuoto. Muistivuoto syntyy kun varattua muistialuetta ei enää käytetä, mutta sitä ei myöskään muisteta palauttaa takaisin allokaatiovarantoon. Kun pääsilmukka suoritetaan uudelleen se kutsuu koodia, joka tekee saman allokaation uudestaan ja prosessin varaaman muistin määrä kasvaa rajatta. Kyseessä on pitkälti käytetystä ohjelmointiympäristöstä johtuva tapaturmainen ongelma. Toisaalta on muistettava, että muistivuotoja on varsin helppo saada aikaa huolimattomalla ohjelmoinnilla myös ympäristöissä, jotka toteuttavat automaattisen muistinhallinnan eli niin sanotun roskien keruun. Esimerkiksi Java-ympäristöön tehdyt ohjelmat unohtavat usein olioita tietorakenteisiinsa, jolloin roskienkeruualgoritmi ei ymmärrä niitä tarpeettomiksi ja syntyy täsmälleen sama tilanne kuin perinteisessä muistivuodossa. Välityspalvelimessa havaitut muistivuodot olivat vähemmän kriittisiin tietorakenteisiin liittyviä, kuten viestien jäljitykseen, lokikirjoitukseen ja tilastojen keruuseen.

Merkittävin muistivuoto oli eräässä vikatilanteessa ilmennyt vuoto. Hallintaviesti saattoi poistaa palvelimen käytöstä silloin, kun useat tilakoneet odottivat palvelimen vapautumista. Tämä on tyypillinen tilanne silloin, kun palvelin on vikaantunut ja valvontaohjelmisto joutuu poistamaan sen. Tällöin palveluviestien tilakoneet eivät koskaan edenneet ja jäivät muistiin haamuna vaikka palvelin olisikin hetkeä myöhemmin rekisteröitynyt uudelleen. Välityspalvelin nimittäin hävitti palvelinta poistaessaan viitteen kaikkiin niihin tilakoneisiin, jotka odottivat palvelimen vapautumista. Muistivuoto tapahtui kuitenkin harvoin ja huomattiin vasta useiden kuukausien ajoajan ja useiden palvelimien vikaantumisten jälkeen. Vuodolla ollut mitään merkitystä järjestelmän toiminnalle, mutta se kuitenkin korjattiin.

Monet muistivuodot etsittiin käyttämällä Newtonin haun sovellusta, jossa sijoitettiin muistiallokaation eron lokille kirjoittava funktio ennen ja jälkeen funktion kutsun, jonka ei olisi pitänyt vuotaa muistia. Muistivuodon löytyessä toinen kutsu sijoitettiin noin puoleen väliin suorituspuuta ja ohjelma ajettiin uudestaan. Näin iteroimalla päädyttiin 3-7:llä yrityksellä osioon, jossa vuoto tapahtui ja joka oli tarpeeksi lyhyt käsin tehtävään tarkastukseen. Muistivuodot ovat vaikeita löytää pelkästään koodin katselmoinnilla, koska ne liittyvät usein virheeseen järjestelmälogiikassa, joka ei ole nähtävissä katselmoinnissa.

Listatietorakenteet osoittautuivat yllättävän vaikeiksi tehdä toimimaan oikein kaikissa tilanteissa, erityisesti olioiden poistosta tuntui löytyvän paljon virheitä. Olisi ollut hyvin järkevää käyttää valmiita tietorakenteita ja jättää tekemättä omat versiot.

Testauksessa tuli esille melko pian ongelmia viestin jäsennyksessä, jonka toteutus ei pystynyt käsittelemään viestejä, joissa oli sisällä merkkejä joita se ei tuntenut. Nämä oli onneksi helppo korjata, joskin niiden löytämiseen meni runsaasti aikaa.

Toteutuksen yhteydessä huomattiin, että välityspalvelin lopetti joskus yhteyksien hyväksymisen ja se jouduttiin käynnistämään uudelleen. Ongelma ei tuntunut korreloivan minkään muun toiminnon kanssa ja joskus vikaantuminen tuntui alkavan itsestään vaikkei välityspalvelinta ei edes käytetty mihinkään pitkään aikaan. Vika tuntui entistä oudommalta, kun se tuntui tapahtuvan vain HP-UX alustalla muttei ei Linux-ympäristössä. Ongelma ratkesi kun välityspalvelimen prosessia tutkittiin "gdb"-perkaimella sen ollessa vielä ajossa. Pian kävi selväksi, että ajastetun lokikirjoitus kirjoitti erääseen väliaikaiseen ja vakiomittaiseen puskuriin muutaman tavun liikaa. Puskuri oli sattunut sijoittumaan HP-UX alustalla saman tietorakenteen viereen, johon vastakkeiden kahvat on talletettu "select"-systeemikutsua varten. Ensimmäisen vastakkeen kahva oli aina palvelinvastakkeen kahva, nyt ajastettu lokikirjoitus virheellisesti kirjoitti nollia näihin tavuihin. Systeemikutsu sai siten arvon 0 ja palautti siten konsolin tapahtumia, eikä palvelinvastakkeen. Kahva 0 tarkoittaa standardia ulostuloa, (eng "stdout") joka ei koskaan voi olla lukuvalmis, tämähän oli tapahtuma jota palvelinvastakkeen kahvalle odotettiin. Näin välityspalvelin ei pystynyt vastaanottamaan uusia yhteyksiä, mutta pystyi palvelemaan hienosti vielä aktiivisia sovelluksia, koska näiden vastakkeet olivat myöhemmin tietorakenteissa, eikä niitä ylikirjoitettu. Linux-alustalle ongelmaa ei esiintynyt, sillä muisti oli

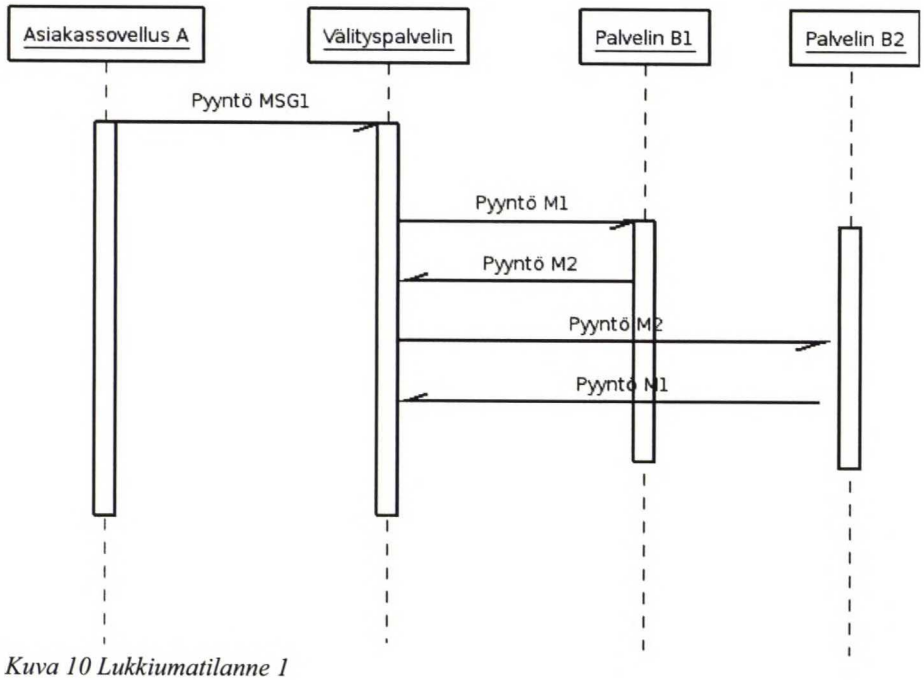
varattu eri tavalla ja lokitustoiminnon ylimääräiset tavut kirjoittivat jonkin muun muistin päälle, mikä ei kuitenkaan ollut kriittistä ohjelman toiminnan kannalta.

Lukuisia muita muistialueiden ylikirjoituksesta johtuvia ongelmia löydettiin, mutta ne johtivat yleensä välittömästi muistialueen saantivirheeseen (eng. "access violation") ja käyttöjärjestelmä lopetti välityspalvelimen prosessin. Muistivedostiedostoista oli helppoa jälkeenpäin selvittää missä ongelma tapahtui ja missä tilassa prosessi juuri silloin oli.

Versionhallinnan laiminlyönti osoittautui virheeksi. Alun perin katsottiin, että versionhallintaa ei tarvita, kunhan vain otetaan säännölliset varmuuskopiot. Käytännössä ongelmia tuli aina, kun yritettiin tutkia toimiko joku ominaisuus edellisessä versiossa, ja missä vaiheessa jokin virhe oli tullut ohjelmistoon. Ilman selkeää versionhallintaa tämä oli vaikeaa, varsinkin kun ei ollut käytössä työkaluja kahden eri version vertailuun.

Eniten toteutusta muuttava ongelma oli täysin ennalta näkemätön ja kuitenkin pohjimmaltaan hyvin yksinkertainen lukittumatilanne. Jälkeenpäin katsoen ongelma on ilmeinen transaktiopohjaiseen viestinvälitykseen perustuvassa arkkitehtuurissa. Kyseessä on syklinen riippuvuus palvelinten välillä, joka aiheuttaa lukkiuman. Lukkiuma tapahtui kahdessa erilaisessa tilanteessa.

Ensimmäinen mahdollinen tapaus on esitetty kuvassa 10.



Kuva 10 Lukkiumatilanne 1

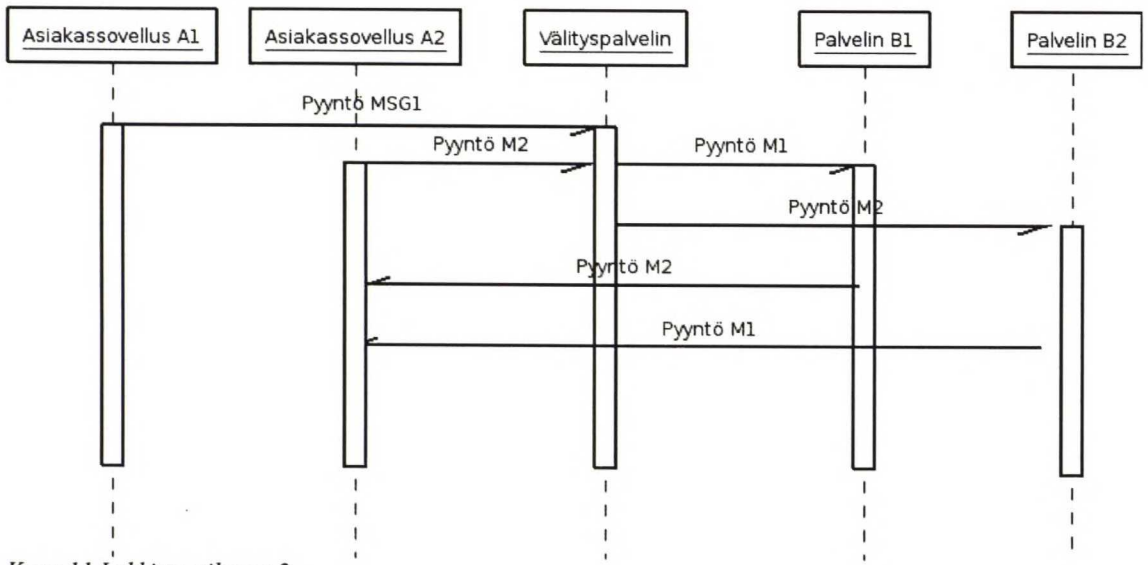
Vaiheiden selitys

1. Asiakassovellus A lähettää viestin M1 välityspalvelimelle, joka ohjaa sen palvelimelle B1
2. Palvelin B1 ryhtyy palvelemaan viestiä ja sitä käsitellessään se lähettää viestin M2.
3. Välityspalvelin ohjaa viestin M2 palvelimelle B2.
4. Palvelin B2 ryhtyy palvelemaan viestiä ja sitä käsitellessään se lähettää viestin M1.

Näin saadaan aikaan lukittuminen, koska B1 riippuu B2:stä ja B2 riippuu B1:stä. Lukkiuman todennäköisyyttä voidaan vähentää käynnistämällä kaksi instanssia palvelimesta B, joskaan se ei poista

ongelmaa. Tämä lukittuminen on deterministinen eikä se riipu prosessien ajastuksesta, se voidaan aina toistaa lähettämällä viesti M1.

Toinen vastaava lukittuminen tapahtui kuvassa 11 esitetyssä tilanteessa.



Kuva 11 Lukkiumatilanne 2

Vaiheiden selitys

1. Asiakassovellus A1 lähettää viestin M1 välityspalvelimelle, joka ohjaa sen palvelimelle B1.
2. Asiakassovellus A2 lähettää viestin M2 välityspalvelimelle, joka ohjaa sen palvelimelle B2.
3. Palvelin B1 ryhtyy palvelemaan viestiä ja sitä käsitellessään lähettää viestin M2.
4. Palvelin B2 ryhtyy palvelemaan viestiä ja sitä käsitellessään lähettää viestin M1.

Lukkiuma syntyy koska kumpikaan palvelin ei pääse etenemään. Lukittuminen on samanlainen kuin edellinen tilanne, mutta riippuu ajasta. Sovelluksen A1 ja A2 pitää lähettää viestit lähes yhtäaikaisesti, että saadaan aikaan kyseinen lukkiuma. Ongelma ei siten ole täysin deterministinen, vaan riippuu käyttöjärjestelmän prosessien ajastuksesta.

Lukkiuman havaitsemiseksi välityspalvelimeen toteutettiin matriisiin perustuva havaintomenetelmä. Ratkaisua tehdessä ei oltu tietoisia mistään lähteestä, jossa ratkaisu olisi esitetty, vaan se on keksitty enemmän tai vähemmän riippumattomasti.

Matriisiin merkitään palvelimet ja niitä palvelevat viestit. Jos matriisissa on kohdassa (S2,M2) alkio, se tarkoittaa, että kyseinen palvelin palvelee viestiä. Oheisessa matriisissa, joka on esitetty taulukossa 12, on esimerkiksi palvelin "S1", joka palvelee viestejä "M1","M3","M4" sekä palvelin "S4", joka palvelee viestiä "M4".

	M1	M2	M3	M4
S1	*		*	*
S2		*		
S3	*			
S4				*

Taulukko 12Palvelumatriisi

Kun palvelin varataan jonkin viestin käyttöön, merkitään palvelumatriisiin palvelimen riville kyseisen viestin kolumniin, että palvelin on varattu tälle viestille. Samalla taulukkoon merkitään mikä on asiakassovellus, joka viestin lähetti. Kuten mainittua, myös palvelin voi lähettää viestejä, joten se voi olla merkittynä palvelimen asiakkaaksi matriisiin. Taulukossa 13 on esitetty esimerkkinä tilanne, jossa asiakassovellus A1 on lähettänyt viestin M2, joka on ohjattu palvelimelle S2. Palvelin S2 on lähettänyt viestin M4, joka on ohjattu palvelimelle S4. Palvelin S4 on lähettänyt viestin M1, joka on ohjattu palvelimelle S3.

	M1	M2	M3	M4
S1	*		*	*
S2		A1		
S3	S4			
S4				S2

Taulukko 13 Matriisin Täyttymisesimerkki

Aina kun viestiä reititetään, voidaan matriisista tarkistaa johtaako viestin palvelu lukkiumaan. Lukkiuman havainnointialgoritmi on esitetty listauksessa 8, missä S on palvelin johon käsiteltävänä oleva viesti on reititetty. Algoritmi palauttaa totuusarvon, joka määrittelee aiheuttaako viestin lähetys palvelimelle lukkiumatilanteen. Algoritmin toteuttavaa funktiota kutsutaan välityspalvelimen toteutuksessa aina reitityksen jälkeen, ennen kuin viesti lähetetään.

```
1.  procedure BOOL deadlock(A,S')
2.  begin
3.    if |ST(A)| > 0
4.    then
5.      for each s in ST(A)
6.      do
7.        if s == S'
8.        then
9.          return TRUE
10.       else
11.         return deadlock(s,S')
12.       fi
```

```
13.     done
14.   fi
15.   return FALSE
16.end
```

Listaus 8 Lukittumisen havainnointialgoritmi

Algoritmi yrittää etsiä sykliä palvelin käytössä. Algoritmissa A on palvelin joka on lähettänyt viestin, S' on palvelin jolle viestiä ollaan reitittämässä. Algoritmissa käytetty apufunktio ST(x) palauttaa rivin, eli vektorin matriisista. Esimerkiksi jos transaktioiden tilanne on taulukossa 14 esitetty ja palvelin S3 yrittää lähettää viestiä M2 syntyy lukkiuma, koska ainoa palvelin joka voi viestiä M2 palvella on S2. Tämä havaitaan algoritmilla seuraavasti. Funktiota "deadlock" kutsutaan argumenteilla A=S3 ja S'=S2. Algoritmi iteroi ja kutsuu itseään argumenteilla A=S1 ja S'=S2. Tällä kertaa iteraatiossa rivillä 7 huomataan lukkiuma koska palvelimen S1 rivillä on merkittynä viestille M4 palvelin S2. Tässä tilanteessa sovellus A1 on lähettänyt viestin M2, joka on ohjattu palvelimelle S2. S2 on lähettänyt viestin M4 joka on ohjattu palvelimelle S1. S1 on lähettänyt viestin M1, joka on ohjattu palvelimelle S3. Nyt palvelimen S3 yrittäessä lähettää viestiä M2 tuloksena olisi esitetty lukkiuma.

	M1	M2	M3	M4
S1	*		*	S2
S2		A1		
S3	S1			
S4				*

Taulukko 14 Lukkiuman Havainnointi

Lukkiuma havaittuaan välityspalvelin ottaa listasta toisen samaa viestiä kuuntelevan palvelimen jos tällaista on saatavilla, ja tarkistaa lukituksen uudestaan. Mikäli toista palvelinta ei löydy, lukitus hoidetaan lähettämällä virhevastaus pyyntöä tekevällä palvelimelle ja kirjoittamalla reitittimen lokille virheilmoitus. Viestien merkitseminen matriisiin ei ole merkityksellistä lukkiuman estämiseksi, mutta siitä on paljon apua lokiviestissä, jonne lukkiumaketju kirjoitetaan sen löydyttyä. Lokiviestin avulla on helppoa analysoida, mitkä palvelimet aiheuttivat ongelman.

Valvontaohjelmiston käytössä huomattiin hyvin pian vakava ongelma, yksinkertainen prosessin käynnistämiseen käytetty suorituskomento "exec"-systeemikutsu ei ollutkaan riittävä. Ongelma johtui siitä, että tällä tavalla käynnistetty uusi prosessi perii isäprosessinsa, eli valvontaohjelmiston, ympäristön. Järjestelmän palvelimet, joita valvontaohjelmiston piti käynnistää, vaativat kuitenkin täysin saman ympäristön kuin käyttäjät, joiden omistajina ne piti käynnistää. Ongelmaan ei oltu varauduttu ja ei ollut tiedossa kuinka prosessi pitäisi käynnistää, että se luulisi olevansa ajossa käyttäjän istunnossa. Ratkaisuksi osoittautui Tatu Ylösen SSH-ohjelman vapaasti saatavilla oleva referenssitoteutus [SSH], jota tutkimalla ymmärrettiin, kuinka istuntoprosessit pitää luoda. SSH-ohjelman lisenssi olisi ilmeisesti sallinut vapaan kopioinnin, mutta siinä katsottiin olevan joitain epäselvyyksiä. Epäselvyyksien vuoksi, ja koska koodi ei muutenkaan ollut täysin tarkoitukseen sopiva, toiminnallisuus päätettiin toteuttaa tyhjästä. Käytetyt systeemikutsut löytyivät kuitenkin tästä esimerkkinä käytetystä toteutuksesta.

Sovellusliittymän toteutuksen yhteydessä yllättäväksi ongelmaksi muodostui kääntäjien murteet. Microsoft Windows Visual Studio C++ kääntäjä ei ymmärtänytkään täysin samaa ohjelmakoodia kuin HP-UX alustan kääntäjä. Ongelmia tuli muun muassa enumeraatioiden, luokkien perinnän sekä staattisten olioviittausten käytössä. Kiusallista oli, että nämä ongelmat havaittiin vasta kun suurin

osa sovellusliittymästä oli kirjoitettu HP-UX kääntäjälle. Näin jouduttiin vaivalliseen uudelleenkirjoittamiseen (eng. "refactoring") joidenkin luokkien osalta. Perimmiltään ongelman aiheutti liian "hienojen" rakenteiden käyttö, kuten moniperinnän.

6.6 Yhteenveto

Arkkitehtuuri osoittautui yllättävän toimivaksi ja sen yksinkertainen rakenne auttoi jälkeenpäin merkittävästi ongelmien selvityksessä. Erityisen hyödyllinen oli tilakonemalli, joka mahdollisti muistivedostiedostojen tutkimisen ja niiden avulla vikatilanteiden deterministisen toiston.

Käytetty ohjelmointikieli osoittautui tarpeeksi hyväksi, joskin sille ei ollut käytännössä mitään vaihtoehtoja. C/C++ toteutuksia vaivaavat virheet, jotka johtuvat alkeellisista muistipalveluista. Muistivuodot ja muistisuojauksen puutteesta johtuvat virheet vievät aina suuren osan ohjelmiston kehitykseen menevästä ajasta.

Ongelmat joita ei nähty ennalta olivat pahimpia, koska niiden takia jouduttiin palaamaan takaisin suunnitteluvaiheeseen ja yleensä muuttamaan joitain jo valmiita osioita. Loppujen lopuksi näitä yllätyksiä oli kuitenkin vähän. Vaikeimmin selvitettäviä ongelma ovat ne, joita ei voida toistaa millään tunnistettavilla ehdoilla, toteutuksen yhteydessä ei onneksi esiintynyt montaa tällaistä ongelmaa.

Viestin jäljitystoiminnasta oli alusta pitäen paljon hyötyä jo pelkästään siksi, että sen avulla oli helppo sijoittaa suoritustauko (eng. "break point") perkaimella vianetsinnän yhteydessä. Itse jäljitystiedostoilla ei ollut käyttöä välityspalvelimen toteutuksen yhteydessä muutoin kuin regressio-testipaketin luonnissa.

Toteutuksen yhteydessä jouduttiin ongelmatilanteissa etsimään uusia ratkaisuja, sekä suunnittelemaan ja toteuttaman nämä ratkaisut. Tämä työ ei mene koskaan hukkaan vaan kokemuksen voi siirtää aina seuraavaan projektiin. Toteutus selkeytti monia asioita kuinka ohjelmisto pitää tehdä, suunnitella ja miten sitä ei pidä tehdä.

7 Testaus

Kappaleessa esitellään välityspalvelimen testausta, testauksen painopistettä ja tärkeimpiä ratkaisuja. Testausta tehtiin jatkuvasti kehityksen ohella rakentaen samalla regressiotestipakettia, joten projektissa ei ollut mitään varsinaista testausvaihetta muutoin kuin suorituskyvyn mittaamiseksi. Tuotanto-ongelmien löytyessä palattiin joskus luomaan tai muuntelemaan testiohjelmia, nämä ongelmat on kuitenkin käsitelty omassa kappaleessaan.

Välityspalvelimen toiminnallisuus on verraten yksinkertaista, joten testitapauksia ei tarvittu kovinkaan paljon. Testauksessa suurimmat haasteet olivat poikkeustilanteissa, eli tilanteissa jossa osapuolet eivät toimineet odotetulla tavalla. Näitä poikkeustilanteita ovat muun muassa ylikuorma, palvelinten vikaantumiset, vikaantuva asiakassovellus sekä verkko-ongelmat. Poikkeustilanteiden tutkimiseksi tehtiin muutama testiohjelma, jotka osasivat emuloida joitain tavallisimpia tilanteita, kuten ylisuuria viestejä sekä liian hidasta palveluvastausta.

7.1 Suunnitelma

Testaussuunnitelma sisälsi vaatimuksista johdetut testitapaukset, mutta näistä pystyttiin johtamaan vain vähän testitapauksia. Koska vaatimukset määrittelivät suhteellisen suppean testijoukon, päädyttiin iteroivaan testaukseen, jossa välityspalvelinta pystyttiin ajamaan jatkuvasti testausohjelmia varten. Välityspalvelimen testiajo ei kuitenkaan ollut tarkoitus olla jatkuvasti ajossa.

Ohjelmistojen testaus on perinteisesti noudattanut vesiputousmallia [Tamres 02], jossa testaus tapahtuu vasta koko järjestelmän ollessa valmis. Tämä ei kuitenkaan ollut tarkoituksenmukaista välityspalvelimen suhteen. Järjestelmän kehityksen piti alkaa melko nopeasti ensimmäisten prototyyppien jälkeen. Sovellusohjelmoijat tarvitsivat vähintään sovellusliittymät päästäkseen aloittamaan järjestelmän toteutukseen.

Vaatimuksissa mainittu testausohjelma tarvittiin sovellusliittymän, välityspalvelimen sekä itse järjestelmän testausta varten. Sovellus päätettiin toteuttaa samalla alustalle kuin valvontaohjelmisto. Microsoft Windows-pohjainen käyttöliittymä sallisi helpon viestin analysoinnin ja tiedon siirron kehittäjien välillä sähköpostilla leikkaa- ja liimaa toiminnallisuutta käyttäen.

Poikkeustilanteiden testaamiseksi päätettiin toteuttaa testisovelluksia. Näitä olivat asiakassovellukset ja palvelinsovellukset, joita pystyttiin parametrisoimaan jossain määrin. Sovellukset käyttivät kuitenkin pelkästään sovellusliittymää, joten testauksessa oletettiin että välityspalvelimeen ei ota yhteyttä sovellus, joka ei käytä viestiprotokollaa määritellyllä tavalla. Pelkästään sovellusliittymään perustuvat testaussovellukset eivät olisi olleet riittävä vaihtoehto, jos jonkin muun osapuolen olisi pitänyt toteuttaa viestiprotokollan toteuttava sovellusliittymä. Tässä tapauksessa käytetty musta laatikko-malli [Tamres 02] katsottiin täysin riittäväksi.

7.2 Toteutus

Testaussovellusten teko oli hyvin helppoa käyttäen valmiita liittymäkirjastoja, lukuisia virheitä saatiin kiinni jo sovellusten tekovaiheessa. Sovellukset toteutettiin sekä Microsoft Windows- että HP-UX alustalle. Sovellukset eivät olleet erityisen joustavia, joten käytännössä eri testitapaukset vain kirjoitettiin sovellusten ohjelmakoodiin. Testaussovelluksilla ei ollut mitään konfiguroitavia testitapauksien kuvaustiedostoja. Käytännössä malli toimi kuitenkin hyvin, koska testitapauksia ei ollut paljon.

Suorituskyky testattiin parametrisoimalla palvelinsovellukset siten, että ne kaiuttivat viestejä takaisin. Asiakassovelluksen lähettäessä viestin se sai saman viestin takaisin ja tarkisti että sisältö on sa-

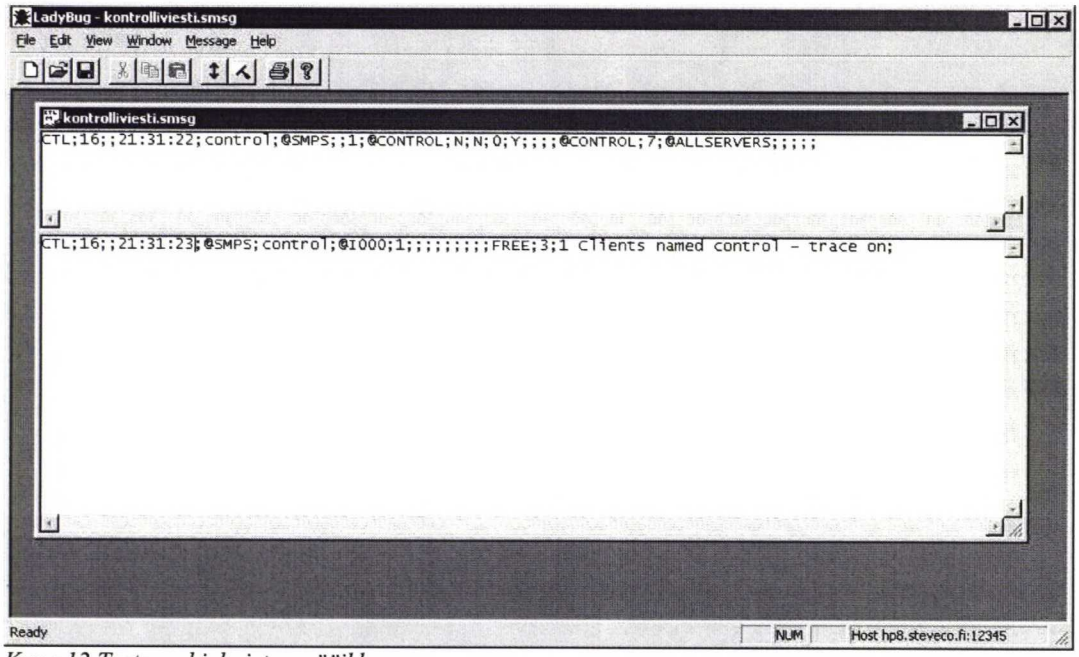
ma. Suorituskyky testattiin sekä HP-UX alustalla, että Linux-alustalla. Koska HP-UX palvelinta tarvittiin jatkuvasti kehityskäyttöön suorituskyky mitattiin Linux-työasemalla, jonka prosessoriteho oli selkeästi alhaisempi kuin HP-UX palvelimen.

Kolmella asiakassovelluksella ja kahdella palvelinsovelluksella, käyttäen kuutta eri viestiä, päästiin suorituskykyyn 100 transaktiota sekunnissa Linux-työasemalla. Viestien keskimääräinen koko palvelimelle oli 200 tavua ja palvelimelta vastausviesti asiakassovellukselle 2 kilotavua. Palvelimen prosessori oli 300Mhz Intel Pentium III. Tuloksien katsottiin riittävän, koska järjestelmää ei koskaan tultaisi ajamaan tämän suuruisella kuormalla. HP-UX palvelin olisi myös huomattavasti tehokkaampi kuin mittaukseen käytetty kone. Suorituskykytestissä ei löydetty mitään ongelmia, ainoa löydetty poikkeama oli aikaisemmin mainittu virhe Linuxin "accept"-systeemikutsussa. Poikkeama ei kuitenkaan merkinnyt mitään, koska Linux ei ollut tuotanto- tai testausalusta.

Järjestelmän kehityksen aikana palvelimet vikaantuivat usein, joten hallintaohjelman jäljitystoimintoa ja testausohjelmaa tarvittiin usein. Järjestelmä yksikkötestaus (eng. "unit testing") oli kuitenkin riippumatonta välityspalvelimen testauksessa ja sen kautta löydettiin ainoastaan sivulla 57 esitetty lukkiumatila.

7.2.1 Testausohjelmistot

Testiohjelmistot jakautuivat siis kahteen osaan, asiakas- ja palvelinsovellusten toimintaa emuloiviin ja viestien testausohjelmaan. Viestien testausohjelma on varustettu graafisella käyttöliittymällä ja sen avulla voidaan lähettää mikä tahansa viesti ja analysoida vastaus selkokielisenä. On kuitenkin huomattava, että edes tämä ohjelmisto ei voi lähettää raakaa tietoa viestiprotokollaan vaan se kelpuuttaa syötteensä ennen viestin lähetystä. Testausohjelma on esitetty kuvassa 12.



Kuva 12 Testausohjelmiston pääikkuna

Testausohjelmalla voi olla useita eri testitapauksia auki yhtä aikaa, kukin omassa ikkunassaan. Ikkuna jakautuu kahteen osaan, ylemmässä on viesti joka halutaan lähettää ja alemmassa on siihen saatu vastaus. Painamalla transaktio-nappia viesti lähetetään konfiguroidulle välityspalvelimelle ja vastaus saadaan alemaan ikkunaan. Viestiä talletettaessa myös vastaus talletetaan, joskin se yli ajetaan

aina kun transaktionappia painetaan. Talletetusta viestistä on helppo nähdä miten jokin viesti on joskus toiminut. Tässä yhteydessä huomattiin, että viestiprotokollaan olisi pitänyt lisätä otsakesegmenttiin myös päivä pelkän kellonajan lisäksi. Näin testausohjelmiston käyttäjän olisi ollut helppo nähdä milloin esimerkiksi jokin viesti on viimeksi toiminut. Viestit testitapauksiin saatiin käytännössä välityspalvelimen jäljitystiedostoista, niitä kuitenkin jouduttiin muokkaan vielä käsin. Tässä yhteydessä testausohjelmiston ominaisuus kelpuuttaa viestit ennen lähetystä osoittautui hyvin arvokkaaksi.

Palvelinsovellus toteutti pääpiirteissään tavanomaisen palvelimen. Pääsilmukan toteutuksessa oli kuitenkin useita eri alirutiineja, joihin voitiin hypätä sen mukaan mitä komentoriviltä oli annettu argumentiksi. Esimerkiksi yksi alirutiini suoritti "exit"-systeemikutsun lopettaen ohjelman, toinen yritti kirjoittaa osoitteeseen 0 laukaisten muistinsuojauspoikkeuksen (eng. "SIGSEGV") ja kolmas kutsui "sleep"-systeemikutsua jollain hyvin pitkällä ajalla. Näin voitiin simuloida käytännössä kaikki tavallisimmat järjestelmässä tapahtuneet vikatilanteet. Asiakassovelluksen rakenne oli käytännössä identtinen palvelinsovelluksen kanssa sillä erotuksella, että lähetettävä testiviesti pystyttiin antamaan komentoriviltä määritetyllä tiedostolla.

Palvelin- ja asiakassovelluksia ajettiin usein valvontaohjelmistolla, joten sen testaus onnistui samalla ilman mitään erityistä testipakettia. Näin myös se pystyttiin testaamaan totuudenmukaisia olosuhteita ja tavallisimpia vikatilanteita vastaan.

Taulukossa 15 on esitetty tuotetut testausohjelmistot.

Ohjelmisto	Selitys	Huomautuksia
ladybug.exe	Testausohjelmisto viestin lähetystä ja analyysia varten.	Käyttää vain sovellusliittymää.
mtdebug.exe	Asiakastestisovellus	Käyttää vain sovellusliittymää.
mtsrvdebug.exe	Palvelintestisovellus	Käyttää vain sovellusliittymää.

Taulukko 15 Testausohjelmistot

Asiakas- ja palvelintestisovellukset toteutettiin myös Microsoft Windows alustalle.

7.3 Yhteenveto

Sovellusohjelmoijat löysivät usein, ainakin alkuvaiheessa, ongelmia välityspalvelimesta tai sen sovellusliittymästä. Näiden ongelmien selvitykseen meni paljon aikaa, joten vaikka sovellusohjelmoijat myös tavallaan varmistivat välityspalvelimen ja sitä kautta koko järjestelmän laatua, tämä ei ollut tietenkään tavoiteltu tilanne. Yllättäen suurimpia ongelmia aiheuttivat sovellusliittymässä käytetyt perustietorakenteet, kuten listat, joista tuntui löytyvän jatkuvasti ongelmia. Testauksen yhteydessä kävi ilmi että näiden toteutuksen haasteet oli selkeästi aliarvioitu. Itse viestien välityksessä löytyi vain vähän ongelmia.

Sovellusliittymä muuttui testausohjelmistoa rakennettaessa, kun huomattiin siinä jotain kömpelöjä ratkaisuja. Laatu parani myös näin välillisesti, koska selkeämpi rajapinta helpotti sovellusohjelmoijien työtä ja sitä kautta koko järjestelmän laatua.

Testauksen ongelma alusta pitäen oli, että se ei ollut kovinkaan säännönmukaista. Testauksen käyttäminen on ehkä liian vahva sana, vaan pitäisi puhua "kokeilusta". Käytännössä tämä kuitenkin riitti, koska sovellusliittymät eivät sallineet suuria poikkeamia siihen, kuinka sovellukset käyttäytyivät. Samaten työmäärä joka käytettiin vikatilanteiden testaamiseen maksoi itsensä takaisin. Tuotanto-ongelmia käsiteltäessä tulee ilmi asioita jotka olisi yksinkertaisesti pitänyt löytää jo testauksen yhteydessä.

Viestien helppo luettavuus osoittautui hyvin arvokkaaksi ominaisuudeksi sekä testauksen että sovelluskehityksen vianselvityksen kannalta. Luettavat viestit auttoivat myös tuomaan uusia kehittäjiä sisään prosessiin, koska järjestelmän ydinidea oli konkreettinen, helposti tartuttava ja ymmärrettävä asia. Itse en enää kyseenalaista yksinkertaisuuden ja läpinäkyvyyden periaatetta ohjelmistokehityksessä.

8 Tuotanto

Kappale käsittelee järjestelmän tuotantoon ottoa siinä määrin kuin se on mielekästä tämän työn kannalta. Käsittelemättä jätetään projektin aikataulutus ja muun järjestelmän aiheuttamat muutokset tuotantoon ottoon. Kappaleessa keskitytään tuotannossa havaittuihin välityspalvelimen tai sovellusliittymän ongelmiin. Järjestelmän tai projektin omia ongelmia ei käsitellä, elleivät ne vaikuttaneet suoraan tai välillisesti välityspalvelimeen. Tämän vuoksi kappaleessa esitetty järjestelmän käyttöönoton kuvaus näyttää todellisuutta kitkattomammalta.

Tuotantoon mentäessä välityspalvelin oli käytännössä täysin valmis, eikä siihen enää tarvittu mitään kehitystä. Yleensäkin tuotantoon oton yhteydessä ei ollut minkäänlaista viime hetken painetta välityspalvelimen toteutuksessa tai testauksessa.

8.1 Suunnitelma ja Toteutus

Järjestelmä otettiin tuotantoon pienissä paloissa yksi alijärjestelmä kerrallaan. Uusi arkkitehtuuri koestettiin ottamalla välityspalvelin käyttöön pienessä alijärjestelmässä, joka oli tehty laajentamaan jo olemassa olevaa vanhaa järjestelmää. Tämä järjestelmä olisi sitten tarkoitus poistaa myöhemmin käytöstä projektin edetessä. Vanhan järjestelmän asiakassovellukseen oli tehty yksi toiminto viestipohjaisella sanomanvälityksellä siten että käyttäjät eivät huomanneet mitään eroa entiseen toiminnallisuuteen.

Myöhemmin järjestelmä otettiin kokonaisuudessaan käyttöön pala kerrallaan, välityspalvelimen asennusta ei tarvinnut ensimmäisen palvelimen jälkeen enää muuttaa, vaan siihen vain yksikertaisesti rekisteröitiin lisää palvelimia. Hallintaohjelmiston konfiguraatioon lisättiin aina yksi alijärjestelmä ja palvelin kerrallaan lisää. Tuotantoon oton yhteydessä ei ollut harvinaista saada useiden kuukausien yhtämittaisia ajoaikoja välityspalvelimelle.

Palvelinsovellukset oli tehty siten, että ne lähes kaikki pystyivät periaatteessa palvelemaan mitä tahansa viestiä. Palvellut viestit oli annettu komentoriviltä. Tällä tavalla tarvittiin vain muutama erilainen palvelinohjelma, joten järjestelmää pystyttiin muokkaamaan haluttuun tasapainoon. Ominaisuus osoittautui hyvin arvokkaaksi sivulla 57 esitetyn lukkiumaongelman välttämiseksi.

Asiakassovellukset rakennettiin myös pala kerrallaan aina lisäten uutta toiminnallisuutta näkymä kerrallaan. Viestien versioiden välinen alaspäin yhteensopivuus osoittautui erittäin tärkeäksi ominaisuudeksi, koska uusia asiakassovellusversioita voitiin päivittää vain sinne missä niitä tarvittiin. Arkkitehtuuri mahdollisti vakaan ympäristön käyttöönottoa varten.

8.2 Liittyminen valvontaratkaisuihin

Valvontaratkaisuihin liittyminen oli yksinkertaista, välityspalvelimeen ei tehty käytännössä mitään ylimääräistä toiminnallisuutta. Valvontaohjelmiston sähköposti konfiguroitiin osoittamaan järjestelmänvalvojan postilaatikkoon ja ulkoisen valvontajärjestelmän sovellus tarkkaili järjestelmän lokitiedostoja. Hallintaohjelmiston lokitiedostoilta tarkkailtiin palvelinten lopettamisia ja uudelleenkäynnistämisiä. Välityspalvelimen lokilta tarkkailtiin lukkiumatilanteita, joskaan niitä ei enää myöhemmin koskaan syntynyt. Valvontajärjestelmän taustaprosessi raportoi myös virhe-tason lokiviestit valvojalle mutta käytännössä jo muista mittareista nähtiin, kun järjestelmässä oli jotain vialla.

Ratkaisu osoittautui riittäväksi ja useimmiten valvontajärjestelmän valvoja ehti soittaa paikalliselle järjestelmänhallitsijalle ennen kuin järjestelmän käyttäjä oli itse ilmoittanut ongelmista.

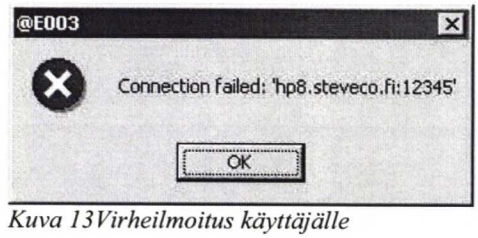
8.3 Tuotanto-ongelmat

Tuotantoon mentäessä välityspalvelimessa ja valvontaohjelmistossa oli jo runsaasti toimintoja vikatilanteiden analyysia varten. Ohjelmistojen lokitiedostot ja valvontaohjelmiston automaattinen sähköpostiviesti kattoivat jo suuren osan vika-analyysia. Käytännössä kuitenkin kävi odotetusti, että suurin osa ongelmista johtui siitä, että järjestelmän palvelimissa oli jotain vikaa. Vian selvittäminen ei tällaisessa ympäristössä kovinkaan helppoa, kun saatavilla on useimmiten vain epämääräinen virheilmoitus käyttäjältä, eikä vika ole enää muutoinkaan toistettavissa. Viestien jäljitys auttaa kyllä paljon vian selvityksessä, mutta vain silloin kun käyttäjä pystyy toistamaan vian ja tekemään sen järjestelmänhallitsijan ohjeita noudattaen. Muutoin jäljitystä ei voida kytkeä oikeaan aikaan päälle.

Tuotannossa kävi siis ilmi, että järjestelmävikaa jouduttiin useimmiten etsimään jälkeenpäin vian lokitiedostot apuna. Siksipä välityspalvelimen lokiin lisättiin ilmoituksen aikarajojen ylityksistä sekä lukkiumatilanteen estosta. Valvontaohjelmiston lokiin lisättiin myös varoituksia kun kyselyviesteihin ei saatu aina odotetusti vastauksia. Tällä saatiin melko nopeasti siedettävä ratkaisu varsinkin kun tehtiin vielä skripti, joka haki päivältä nämä tiedot lokilta ja esitti ne ymmärrettävässä muodossa. Raportista on mahdollista arvioida mitkä palvelimet tai viestit aiheuttivat ongelmia.

Välitysohjelmiston käyttöönoton jälkeen yleisimmäksi ongelmaksi osoittautuivat liian löyhät hakuparametrit, jotka kuormittivat palvelimia liikaa johtaen aikarajan ylitykseen. Toinen erittäin tavallinen ongelma olivat aivan tavanomaiset kirjoitusvirheet, joita käyttäjä ei juuri sillä hetkellä pystynyt huomaamaan. Kirjoitusvirheisiin perustuvat ongelmat ovat erittäin tavallisia tietokoneen käytössä, kun käyttäjä tulee niin sanotusti sokeaksi kirjoittamalleen tekstille. Nämä ongelmat olivat aina toistettavia ja saatiin yleensä helposti välityspalvelimen jäljitystoiminnolla kiinni, kunhan käyttäjällä oli vain aikaa noin 15 minuutin vianselvitykseen.

Käyttäjien vianraportoinnin parantamiseksi tehtiin sovellusliittymään muutoksia. Suurin osa muutoksista tehtiin asiakassovellusten Microsoft Windows-version sovellusliittymään, jossa käyttäjälle pystyttiin esittämään jonkinlainen ymmärrettävä virheilmoitus. Virheilmoitus on yksinkertainen ikkuna, jossa on esitetty virheen koodi ja selite. Esimerkki virheilmoituksesta on esitetty kuvassa 13, jossa on englanninkielinen versio eräästä virheestä.



Kuva 13Virheilmoitus käyttäjälle

Sovellusliittymään sisäänrakennetut virheviestit lokalisoitiin myöhemmin suomeksi, näin käyttäjät muistivat huomattavasti paremmin mikä virhe oli kyseessä. Käyttäjät kuittasivat useimmiten englanninkielisen ikkunan välittömästi pois edes lukematta sitä.

Erääksi puutteeksi havaittiin se, että ei ollut mitään tapaa jolla palvelinsovellus olisi voinut kommunikoida asiakassovelluksen kanssa. Tähän mietittiin ratkaisua, jossa palvelin olisi voinut jättää viestin välityspalvelimeen, joka olisi liitetty seuraavaan vastauksen joka asiakassovellukselle annetaan. Tästä olisi kuitenkin seurannut lisää ongelmia, koska asiakassovelluksen seuraavan pyyntöviestin ajanhetkeä ei voinut tietää etukäteen, eikä voitu tietää mille palvelimelle viesti ohjautuu. Palvelimen olisi myös täytynyt pitää tilaa yllä siitä, kuka oli viimeksi lähettänyt minkäkin pyynnön. Palvelinten asiakassovelluksille lähettämiä ilmoituksia olisi käytetty hyvin pitkäaikaisten palvelupyyntöjen seurantaan, kuten esimerkiksi raporttien generoinnissa, jotka kestivät rutiininomaisesti tunteja. Palvelinten lähettämistä viesteistä asiakassovelluksilla luovuttiin ja ongelma ratkaistiin yksinkertaisella

palvelimella, joka luki kannasta raportoinnin eräajojen tilan. Raporttien eräajojen piti tietysti tallettaa tämä tila kyseiseen tietokantauluun. Asiakassovellukseen tehtiin omaan säikeeseensä alirutiini joka palveli ikkunaa mihin eräajon edistymisaste raportoitiin. Tällä saatiin riittävä palaute asiakassovellukselle raportoinnin edistymisestä. Ikkunaa palveltiin omassa säikeessään, joten tässä yhteydessä jouduttiin tekemään sovellusliittymä säieturvalliseksi. Tämä osoittautui ennakoitua helpommaksi, koska tilattomien transaktiot toteuttava koodi ei tarvitse juurikaan synkronointipisteitä.

Eräajojen käynnistyksessä syntyi ongelmia, kun jotkut käyttäjät käynnistivät lukuisia eräajoja rinnakkain. Tämä vei järjestelmästä kaikki resurssit ja hidasti muiden käyttäjien toimintoja kohtuuttomasti. Valvontaohjelmistoon tehtiin muutos, ettei sama käyttäjä voisi käynnistää useita eräajoja liikaa. Tämä toteutettiin asettamalla yksinkertainen laskuri valvontaohjelmistoon, joka määritteli kuinka monta prosessia ajossa sai kustakin eräajosta olla. Tällä varmistettiin järjestelmän käytettävyys silloinkin kun kaikki käyttäjät laittoivat raportoinnin käyntiin ennen lounastunnin alkua, vaikkeivat olisi edes tarvinneet kyseistä raportti juuri lounaan jälkeen. Tämä oli ainoa merkittävä muutos, joka jouduttiin tekemään ohjelmistoon käyttäjien tottumuksien tukemiseksi.

Ongelma jolle ei löytynyt mitään hyvää ratkaisua, mutta joka aika-ajoin esiintyi, olivat pitkät viiveet palvelimissa. Joskus järjestelmässä vain kävi niin syystä tai toisesta, että palvelin joutui käyttämään runsaasti aikaa jonkin viestin palvelemiseksi ja välityspalvelimen aikaraja ylittyi. Tällöinhän palvelin lopetetaan ja käynnistetään uudelleen. Käytännössä käyttäjät huomasivat ennen pitkää minkälaiset toiminnot olivat herkkiä aiheuttamaan ongelman ja välttivät niitä tai tekivät ne vain aikoina, kun järjestelmän kuorma ei ollut raskaimmillaan. Ongelma käytännössä katosi myöhemmin HP-UX palvelimen päivityksellä.

Toinen liian pitkään palveluaikaan johtava ongelma pystyttiin ratkaisemaan. Aikarajan ylitys johtui usein liian löyhistä hakuparametreista, jotka asiakassovellus hyväksyi käyttöliittymään. Suurin osa palvelinten operaatioista nimittäin redusoitui suoriksi SQL-tietokantaoperaatioiksi, joten nämä aiheuttivat erittäin raskaita hakuja jotka kestivät kauan. Aluksi eliminoitiin kaikkien räikeimmät tapaukset, kuten pelkät "*" ja "?" merkit hakuehtoina, jotka siis saattoivat tarkoittaa mitä tahansa merkkiä. Myöhemmin käyttäjät kuitenkin oppivat tekemään hakuja tyyliin "*?" tai "????", joten taas käyttöliittymää piti muokata älykkäämmäksi. Samaan aikaan jäljitysviesteistä tutkittiin mitä hakuehtoja käytettiin eniten ja tietokantaan luotiin lisää indeksejä nopeuttamaan näiden parametrien hakuja. Lopullinen ratkaisu oli kaksijakoinen. Asiakassovellukseen tehtiin algoritmi joka yritti arvioida kuinka avoin hakuehto oli, jokainen merkki pisteytettiin ja jos hakuehto ylitti kovakoodatun arvon sitä ei joko kelpuutettu tai siitä ainakin varoitettiin. Palvelinsovelluksiin tehtiin esikysely, joka yritti erikoistuneella SQL-kyselyllä varmistaa kuinka monta riviä hausta muodostuisi ennen kuin se ryhtyi palvelemaan viestiä. Tämä oli kuitenkin työlästä ja tehtiin vain pahimmille ongelmaviesteille. Sovellusliittymäkirjaston aikarajan ylityksestä kertovaan virheilmoitukseen lisättiin myös muistutus hakuehtojen paremmasta määrittelystä.

Valvontaohjelmistosta löytyi harvoin esiintynyt virhe joka haittasi raportoinnin eräajojen käynnistymistä. Valvontaohjelman saadessa käskyviestin, se käynnistää eräajoprosessin määritellyllä tavalla, komentorivi prosessin käynnistämiseen tulee viestin mukana. Eräajoprosessi on ajossa aikansa ja sitten lopettaa itse suorituksensa. Kun prosessi lopettaa UNIX-järjestelmässä suorituksensa, käyttöjärjestelmä lähettää isäprosessille "SIGCHILD"-signaalin joka kertoo isäprosessille, että sen pitäisi kutsua "wait"-systeemikutsua saadakseen lapsiprosessin lopetustilan (eng. "exit status"). Jos lopetustilaa ei kysellä jää lapsiprosessi ns. "zombie"-tilaan, eikä sitä saada pois käyttöjärjestelmän prosessitaulusta [Stevens 92]. Zombie-prosessit voivat lopulta täyttää käyttäjälle varatun prosessiavaruuden ja estää uusien prosessien käynnistämisen. Valvontaohjelmisto käyttää aina kyselyviestin saatuaan "prstat"-systeemikutsua, jolla se yrittää kysellä käynnistämiensä prosessien tilaa. Ongelma syntyi kun tätä systeemikutsua kutsuttaessa juuri samalla hetkellä jokin lapsiprosessi lopetti suorituksensa ja käyttöjärjestelmä nosti "SIGCHILD"-signaalin. Tämä johti siihen, että "prstat"-systeemikutsu palasi virhekoodilla "EINTR" joka tarkoittaa keskeytettyä systeemikutsua (eng. "interrupt-

ed system call"). Tähän ei kuitenkaan oltu varauduttu eikä sitä oltu koskaan huomattu testauksessa koska kyseisen systeemikutsun vaatima aika on erittäin lyhyt. Näiden tapahtumien yhtäaikainen suoritus on hyvin epätodennäköistä. Valvontaohjelmisto luuli että kyselty prosessi ei ollut käynnissä ja joutui näin ollen vikatilaan, kun sen näkemys prosessin tilasta ei enää vastannut todellisuutta. Ongelma ratkaistiin lisäämällä tilanteelle erikoiskäsittely ja välitön uudelleenyritys.

Tuotannossa huomattiin ongelma joka olisi ehdottomasti pitänyt huomata jo testauksessa, kävi nimittäin ilmi, että useampi palvelin pystyi rekisteröitymään tyhjällä nimellä välityspalvelimeen. Tämä ei onnistunut silloin, kun palvelimilla oli edes joku nimi, vaikka tyhjä välilyöntimerkki. Kumpikaan tilanne ei ollut oikea, joten välityspalvelin korjattiin pikaisesti.

Lokitiedosto osoittautui ongelmalliseksi, se nimittäin kasvoi rajatta ja aina samalla nimellä, joten sitä ei pystynyt poistamamaan ilman välityspalvelimen uudelleenkäynnistämistä. Välityspalvelimen lokikirjoitus korjattiin siten, että se kirjoittaa jokaiselle päivälle oman lokitiedostonsa eikä vikaannut jos tiedosto on kirjoitusten välillä poistettu. Tässä tilanteessa se vain luo tiedoston uudelleen.

Tilakoneessa havaittiin ongelma jossa se joskus kirjoitti palvelinten vastaukset asiakassovelluksille rikkinäisinä. Jäljityksessä ei kuitenkaan havaittu mitään ongelmaa, vaan siellä viestit olivat täysin ehjiä. Ongelma esiintyi vain kovassa kuormassa, eli silloin kun HP-UX palvelimen koko prosessoriteho oli käytössä. Verkkokaapparilla tutkittaessa nähtiin, että viesteistä puuttui usein mielivaltainen palanen tietoa, eikä tämä palanen yleensä koskaan ollut esimerkiksi yksi puuttuva segmentti. Jäljitystiedostoissa oli kuitenkin täysin oikeamuotoinen viesti, joten ongelma oli ilmeisesti siinä, kuinka tilakonefunktio kirjoittaa viestiä asiakassovellukselle tilassa "CLIENT_WRITE". Tila on esitelty tarkemmin taulukossa 5 . Ongelma selvisi lisäämällä lokikirjoitusta viestin kirjoittamiseen, josta havaittiin, että kovassa kuormassa "write"-systeemikutsu, joka kirjoittaa tiedon TCP-puskuriin, ei aina saanut kirjoitettua kaikkea tietoa kerralla. Tilakone ei tarkistanut oikein systeemikutsun paluuarvoa ja näin se välillä hukkasi sisäisestä puskuristaan tavuja. Sitä vastoin jäljityksessä sama systeemikutsu onnistui aina, koska se kirjoitti tavut paikalliseen tiedostoon. Ongelma ratkaistiin korjaamalla tilakoneen tilafunktion toteutus ja samalla korjattiin myös virheitä tiedon puskuroinnissa siltä varalta, että ongelma olisi esiintynyt myös muissa yhteyksissä. On huomattava, että vika olisi voinut esiintyä missä tahansa kirjoittavassa tilassa, mutta se satuttiin havaitsemaan vain yhdessä.

Hitaiden yhteyksien päässä käytetyt sovellukset epäonnistuivat joskus viestin vastaanottamisessa, saatu virhekoodi oli "zlib"-kirjaston antama integriteettivirhe. Ongelma toistui harvoin, eikä se tuntunut haittaavaan millään tavalla järjestelmän käyttöä. Aluksi epäiltiin, että kyseisessä työasemassa oli viallinen muistikampa tai vastaava laitepohjainen vika. Kerran kävi kuitenkin niin, että eräs toiminto aiheutti toistettavasti saman virheen. Käytännössä tämä tarkoitti sitä, että jos käyttäjä syötti tietyn kontin numeron käyttöliittymän lomakkeeseen oheistietoineen ja painoi nappia virheikkuna hypähti näytölle. Järjestelmänvalvoja laittoi käyttäjälle jäljitystoiminnon päälle ja näin ongelman aiheuttava viesti saatiin talteen. Kävi ilmi, että palvelimen vastausviesti oli pakatussa muodossa sellainen, että siinä oli erikoinen yhdistelmä koodinvaihtomerkkejä. Tämä aiheutti viestin jäsentäjän virheellisen toiminnan, joten pakkaustoteutus sai rikkinäistä tietoa. Oli kuitenkin vielä selvitettävä miksi vasta nyt virhe oli ollut toistettava, sillä kyseessähän oli täysin deterministinen ongelma. Syyksi paljastui joissain viesteissä käytetty kellonaika, joka oli joka kerta muuttanut paluuviestiä. Käyttäjä yritti viestiä usein hetken päästä uudelleen ja kun virheen jälkeen muutama sekunti oli kulunut, ei vastausviesti ollut enää pakattuna samanlainen ja sen jäsennys onnistui. Käyttäjät olivat myös oppineet, ilmeisesti vahingossa, että ylimääräisen välilyönnin lisääminen johonkin viestiin liitettävän käyttöliittymän kenttään eliminoi ongelman. Tässä toistettavassa tapauksessa kumpikaan ehto ei ollut täyttynyt.

Vuosi käyttöönoton jälkeen huomattiin, että kun välityspalvelin oli ollut ajossa noin 6 kuukautta yhtäjaksoisesti, sen varaaman muistin määrä oli kasvanut useisiin kymmeniin megatavuihin. Tarkem-

min tutkittaessa havaittiin että ongelmana oli palvelimien vikaantumiseen liittyvä muistivuoto. Tätä on käsitelty tarkemmin sivulla 56.

9 Yhteenveto

Lopputuloksena saatiin välityspalvelin, joka kykeni täyttämään sille asetetut vaatimukset ja ehkä jopa jossain määrin ylittämään ne. Järjestelmä saatiin tuotantoon, eikä välityspalvelimesta löytynyt sellaisia ongelmia, jotka olisivat haitanneet merkittävästi järjestelmän käyttöä. Sovellusliittymä välityspalvelimeen osoittautui myös käyttökelpoiseksi, eikä siitä tullut pullonkaulaa tai muuta estettä järjestelmää rakennettaessa.

Arkkitehtuuriset päätökset ja suunnittelussa tehdyt oletukset osoittautuivat pääpiirteittäin oikeiksi. Ohjelmiston rakennetta ei tarvinnut enää tuotantovaiheeseen mennessä muuttaa radikaalisti, eikä se aiheuttanut merkittäviä tuotantokatkoksia. Tilakoneisiin perustuva suunnittelu osoittautui myös menestykselliseksi, ohjelmisto pystyttiin toteuttamaan melkein suoraan spesifikaatiosta ilman kuilua vision ja todellisuuden välillä.

Suorituskyky osoittautui täysin riittäväksi. Tuotantoon siirtyminen asteittain tietysti auttoi ongelmien selvityksessä, ennen kuin merkittävä osa järjestelmää olisi riippuvainen välityspalvelimesta. Testaus olisi pitänyt määritellä tarkemmin, nyt oli hieman onneakin matkassa, ettei huomaamatta jääneistä ongelmista tullut vakavia. Ainakin ongelmat pystyttiin nyt korjaamaan, ennen kuin niistä oli merkittävä haittaa.

Sovellusliittymä olisi voinut olla paremmin suunniteltu, kapea kokemus olio-suunnittelussa näkyy siinä selkeästi nyt jälkeenpäin katsottaessa. Tosin myöhemmät vertailut kaupallisten tuotteiden rajapintoihin paljastivat, etteivät nekään olleet kovin selkeitä tai intuitiivisia. Ilmeisesti rajapinnan joustavuus on aina jossain määrin kääntäen verrannollinen sen käytettävyyteen ja ymmärrettävyyteen.

Ajatus, että virheet piilotetaan käyttäjältä mahdollisimman hyvin osoittautui menestykselliseksi, käyttäjiltä ei tullut juuri koskaan negatiivista palautetta. Vanhaa sanontaa mukaillen, hyvä ohjelmisto on sellainen ettei sitä huomaa ennen kuin se vikaantuu.

Viestien luettavuus oli erittäin hyvä päätös määrittelyssä, se auttoi paljon myöhemmissä vaiheissa vianselvityksen ja järjestelmän ymmärrettävyydessä.

Hallintaohjelmistojen rakentaminen ja niiden vaatimusten täyttäminen opettivat hyvin paljon ohjelmistosta kokonaisuutena. Pelkkä hyvin toimintansa suorittava palvelin ei riitä, sitä täytyy pystyä monitoroimaan, konfiguroimaan ilman tuotantokatkoksia ja sen täytyy pystyä auttamaan järjestelmän vianselvityksessä. Valvontaohjelmisto olisi voinut olla huomattavasti yksinkertaisempi, mutta sen rakentaminen nykyiseen muotoonsa opetti jälkeen miten vaikeaa todella luotettavan ohjelmiston rakentaminen on.

Kaikenkaikkiaan projekti oli menestys, vaikkakin siihen meni henkilökohtaista aikaakin niin paljon, että kaiken ajan laskuttaminen olisi tehnyt välityspalvelimen tekemisestä kaupallisesti kannattamatonta. Joka tapauksessa järjestelmä toimii, on tuotannossa ja oli tekijälleen tärkeä oppimiskokemus josta on ollut mittaamatonta hyötyä jälkeenpäin.

Välityspalvelin on nykyään tuotannossa Stevecon Kotkan satamassa PRO2000 järjestelmässä. Järjestelmän elinkaari on ajoitettu pitkälle vuosikymmenen loppuun ja sen yli. Välityspalvelinta ei ole tarkoitus tänä aikana vaihtaa.

Välityspalvelimen Microsoft Windows käännöstä on käytetty joissain kokeellisissa projekteissa, mutta ei tuotantokäytössä.

Tuotantoon lähdön jälkeen Steveco osti laskutusjärjestelmän kolmannelta yritykseltä, joka käytti sitten välityspalvelimen sovellusliittymä integroituessaan PRO2000-järjestelmää. Tämän käyttöönotto onnistui ilman muutoksia välityspalvelimeen tai sovellusliittymään.

Välityspalvelimen valmistuttua kävi melko pian selväksi ettei sitä tulla jatkokehittämään, vaikkakin sen käyttämistä eri projekteissa harkittiin. Tämä johtui lähinnä siitä, että TietoEnator:in strateginen

asema on konsultaatiossa eikä työkaluohjelmistojen tuottamisessa. Välityspalvelin tulee jäämään yhdeksi perinnejärjestelmäksi (eng "legacy system") yhteen järjestelmään, jonka joku muu projekti jokin päivä tulee korvaamaan.

Lähdeluettelo

[Apache] Apache Foundation, URL: http://www.apache.org, Syyskuu 2003

[Appel 97] A. W. Appel, "Modern Compiler Implementation in Java", Cambridge University Press, Joulukuu 1997

[ATK 03] Tietotekniikan liiton sanastotoimikunta, "ATK-SANAKIRJA", 12 painos, 2003

[Balena 99] F. Balena, "Programming Visual Basic 6", Microsoft Press International, Kesäkuu 1999

[BEA] Bea Systems, URL: http://www.bea.com, Syyskuu 2004

[Chap 96] D. Chappell, "Understanding ActiveX and OLE", Microsoft Press; 1st Edition, Tammikuu 1996

[Corm 96] Cormen, Leiserson, Rivest, "Introduction to Algorithms", 16th painos., The MIT Press, 1996

[Fowler 00] M. Fowler, "UML Distilled 2nd Edition - A Brief Guide to the Standard Object Modelling Language", Addison-Wesley, USA 2000

[Gamma 95] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley, USA 1995, sivu 305.

[Hall 96] C. L. Hall, "Bulding Client/Server Applications Using TUXEDO", Wiley, USA 1996

[Java] J. Gosling, B. Joy, Guy Steele, Gilad Bracha, "The Java Language Specification, Second Edition", Addison-Wesley, Kesäkuu 2000

[Linux] Linux Online, URL: http://www.linux.org/, Syyskuu 2004

[LXE] LXE Company, URL: http://www.lxe.com/, Lokakuu 2004

[Naur 60] Naur, Peter, "Revised Report on the Algorithmic Language ALGOL 60.", Communications of the ACM, Vol. 3 No.5, sivu. 299-314, Toukokuu 1960.

[Posix 96] Institute of Electrical & Electronics Engineers, Posix Standard - IEEE Std 1003.1-1996 (POSIX-1)

[Jones 99] R. M. Jones, "Introduction to MFC Programming with Visual C++", Prentice Hall, Joulukuu 1999

[Tamres 02] L. Tamres, "Introducing Software Testing", Addison Wesley, Elokuu 2002

[Tanen 02] A. S. Tanenbaum, "Computer Networks, Fourth Edition", Prentice Hall, Elokuu 2002

[Tanen 93] A. S. Tanenbaum, "Modern Operating Systems". Prentice-Hall, USA 2001, sivu 110

[ISO646] International Organization for Standardization, ISO/IEC 646:1991 International Reference Version, ITU-T Recommendation T.50 (09/92)

[ISO7498] International Organization for Standardization, ISO 7498 OSI Reference Model

[ISO8859] International Organization for Standardization, ISO/IEC 8859-1 Character Set

[ISO9075] International Organization for Standardization, ISO/IEC 9075:1992, Structured Query Language

[ISO14882] ANSI, "Programming Languages – C++ ISO/IEC 14882:2003", Lokakuu 2003

[Marcus 00] E. Marcus, H. Stern, "Blueprints for High Availability", Wiley, USA 2000

[Prim 95] F. Primatesta, "TUXEDO An Open Approach to OLTP", Prentice-Hall, Iso-Britannia 1995

[RFC791] University of Southern California, "INTERNET PROTOCOL"
URL: http://www.ietf.org/rfc/rfc791.txt, Elokuu 2004

[RFC793] University of Southern California, "TRANSMISSION CONTROL PROTOCOL"
URL: http://www.ietf.org/rfc/rfc793.txt, Elokuu 2004

[RFC821] J. B. Postel, "SIMPLE MAIL TRANSFER PROTOCOL"
URL: http://www.ietf.org/rfc/rfc821.txt, Elokuu 2004

[RFC1094] Sun Microsystems, Inc. "NFS: Network File System Protocol Specification",
URL: http://www.ietf.org/rfc/rfc1094.txt, Elokuu 2004

[RFC1951] P. Deutsch, "DEFLATE Compressed Data Format Specification version 1.3",
URL: http://www.ietf.org/rfc/rfc1951.txt, Elokuu 2004

[RFC1738] T. Berners-Lee ja muut, "Uniform Resource Locators (URL)",
URL: http://www.ietf.org/rfc/rfc1738.txt, Elokuu 2004

[RFC2616] R. Fielding ja muut, "Hypertext Transfer Protocol – HTTP/1.1".
URL: http://www.ietf.org/rfc/rfc2616.txt, Elokuu 2004

[Sch 95] B. Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition", Wiley, Lokakuu 1995

[Schm 00] D. Schmidt, M. Stal, H. Rohnert, F. Buschmann, "Pattern-Oriented Software Architecture vol 2 - Patterns for Concurrent and Networked Objects", Wiley, Iso-Britannia 2000, sivu 423

[SSH] SSH Communications Security, URL: http://www.ssh.fi/, Syyskuu 2004

[Stev 01] Steveco Konserni Oy, Vuosikertomus 2001

[Stevens 91] W. R. Stevens, "UNIX Network Programming", Prentice Hall, 1990

[Stevens 92] W. R. Stevens, "Advanced Programming in the UNIX Environment", Addison-Wesley,1992

[TE 2001] TietoEnator Oyj, Vuosikertomus – liiketoimintakatsaus 2001

[WTP] WAP Forum™, "Wireless Transaction Protocol", WAP-224-WTP-200010710-a. Heinäkuu 2001, URL: http://www.openmobilealliance.org/

[ZLIB] J-L. Gailly, M.Adler, "Zlib Compression Library" URL: http://www.zlib.org/, Lokakuu 2004